Proceedings of The 13. Nordic Workshop on Secure IT Systems, NordSec 2008, Kongens Lyngby Oct 9-10, 2008

Nielson, Hanne Riis; Probst, Christian W.

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

Proceedings of
The 13th Nordic Workshop on Secure IT Systems
NordSec 2008

October 9-10, 2008, Kongens Lyngby, Denmark

Hanne Riis Nielson, Technical University of Denmark

Christian W. Probst, Technical University of Denmark

In cooperation with

Microsoft Research

DTU Informatics Graduate School ITMAN

FIRST Research School

DTU Informatics
Department of Informatics and Mathematical Modeling

Technical University
of Denmark

DTU

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-TECHNICAL-REPORT: ISSN 1601-2321

Preface

This volume contains the papers presented at NordSec 2008, the 13th Nordic Workshop on Secure IT-Systems. In 1996 the NordSec workshop series was started as a conference targeting post-graduate students, aiming at bringing researchers and practitioners from industry together. In 2008 we are emphasizing this goal by organizing NordSec as part of the "Nordic Security Days". Other events in this week include a one-day workshop on "Security for the Citizens", where Danish research projects present their results to industry, a meeting of students enrolled in the Erasmus Mundus Master's Programme in Security and Mobile Computing "NordSecMob", and a meeting of the FIRST research school.

We received a total of 39 submissions in response to the call for papers, and the programme committee selected 17 of these for presentation at the workshop—one short paper, which is not included in the proceedings, and 16 research papers. It was a pleasure for us to work with the program committee, and we want to thank both them and the additional reviewers. Beside the submitted contributions we invited Audun Jøsang, University of Oslo, and Michael Huth, Imperial College London, to speak at the workshop. Abstracts of their talks are included in the program.

Finally, we want to thank all who submitted a paper to NordSec.

October 2008 — Hanne Riis Nielson and Christian W. Probst

Program Committee

André Adelsbach	Telindus PSF S.A., Luxembourg
Matt Bishop	University of California, Davis, U.S.A.
Agostino Cortesi	Università Ca' Foscari Venezia, Italy
Mads Dam	Royal Institute of Technology, Sweden
Ivan Damgård	Aarhus University, Denmark
Ulfar Erlingsson	Reykjavík University, Iceland
Viiveke Fåk	Linköping University, Sweden
Dieter Gollmann	Technische Universität Hamburg-Harburg, Germany
Erland Jonsson	Chalmers University of Technology, Sweden
Jan Jürjens	The Open University, Great Britain
Svein Knapskog	The Norwegian University of Science and Technology
Peeter Laud	University of Tartu, Estonia
Hanne Riis Nielson (Co-Chair)	Technical University of Denmark
Kaisa Nyberg	Helsinki University of Technology, Finland
Christian W. Probst (Co-Chair)	Technical University of Denmark
Antti Ylä-Jääski	Helsinki University of Technology, Finland

Additional Reviewers

Magnus Almgren, Tuomas Aura, Billy Brumley, Jani Heikkinen, Jaakko Kangasharju, Ulf Larson, Sven Laur, Janne Lindqvist, Henrik Pilegaard, Vilhelm Verendel, Douglas Wikström

Security Usability
(Invited Talk)

Audun Jøsang
University of Oslo
josang@unik.no

Usability is the weakest link in the security chain of many prominent applications. Poor security usability leads to security vulnerabilities that can be exploited by hackers and criminals. Although the problem of poor security usability was already pointed out by Kerckhoffs in 1883, it currently receives little attention in industry and the research community. The charteristics of security usability are quite different from those of traditional usability. For example, if a computer has poor usability, then people will have trouble getting any useful function out of it. On the other hand, if security protection of the computer has poor usability, then it doesn't neccessarily stop people from from using and getting useful functions out of it. Another interesting difference is that people with trouble using a computer could be ridiculed at worst, whereas people with trouble protecting their computer could be swindled. Sound security usability principles can be defined, but it is often difficult to satisfy such principles in practice. This talk focuses on the cause and consequences of poor security usability, and how to follow principles for good security usability.

Speaker Bio

Audun Jsang joined the University of Oslo and UNIK in March 2008. Prior to that he was Associate Professor at QUT and research leader of the Security Unit at DSTC in Brisbane, Australia, worked in the telecommunications industry for Alcatel in Antwerp, Belgium and for Telenor in Norway. He was also Associate Professor at the Norwegian University of Science and Technology (NTNU). He has a Bachelors degree in telematics from NTH, a Masters in Information Security from Royal Holloway College London, and a PhD from NTNU in Norway.

A Context Aware Network-IDS

Lexi Pimenidis[◇] Benedikt Westermann[†] Volker Wetzelaer[‡]

[◇]University of Siegen, Germany

[†]Center for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology, Trondheim, Norway[*]

[‡]QSC AG, Cologne, Germany

Abstract

In this paper we show that, given appropriate context information, it is possible to reduce the number of false positive errors in an network intrusion detection system.

We discuss technical and organizational means to gather and structure additional information about the hosts' operating system and application. Furthermore, we present an architecture that works on top of an off-the-shelf IDS and demonstrates the feasibility of this approach.

1 Introduction

Nowadays, IT-Systems are threatened by a plethora of attacks. These attacks range from automated worm spreading attempts to targeted attacks; unfortunately the majority of these happen over communication networks. Thus, an important element of a secure network setup is an intrusion detection system (IDS) [2]. The main purpose of an IDS is to inform administrators of a system when their system is under attack. It stores traces of irregular behaviour which can be used in an forensic analysis in the course of incident response actions. In order to achieve this, an IDS needs to have some means of accessing the current network communication (or samples thereof), match it to some known good or bad samples and react accordingly.

An IDS knows neither everything about the system it protects[1] [6] nor about all known, or even unknown, attacks. Thus, it necessarily must make assumptions about the missing information. This, in turn, leads to a certain rate of wrong decisions, namely false positive and false negative reactions. A false positive reaction, also called error of type I, of an IDS is an alert on an attack, which is none. Errors of type II, i.e. false negative reactions happen, if an IDS does not recognise an attack. Due to the nature of type I and type II errors, reducing the number of false positive reactions will usually increase the number of false negative ones, and vice-versa.

Unfortunately, the accuracy of an IDS suffers a lot due to the high complexity of the protected systems and the ever-increasing number of attacks. Due to security reasons, vendors and administrators typically choose to minimize the rate of false negative alarms, therewith resulting in a

[*] *"Center for Quantifiable Quality of Service in Communication Systems, Center of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT.* `http://www.q2s.ntnu.no`

[1] The IDS would need to be a perfect replication of the protected system, in order to do so.

high rate of false positive. As explained above, it is beyond the capabilities of the IDS to further distinguish its final output into correct alarms and false ones. Thus, an system operator has to go through the output of the system in a manual fashion and checks the output for security related incidents. In many cases, and mainly in busy networks, the standard approach leads to a huge amount of alerts generated by the IDS, where the majority is false positive.

On the one hand, false positives alarms by themselves do no direct harm to the security of a system. Getting a large amount of false positive alarms in order to have a only very few false negative alarms seems to be the better trade-off. On the other hand, every alarm has to be manually evaluated by the administrator in order to distinguish false positive alarms from real attacks. Due to this a huge proportion of false alarms results in indirect damage: if the administrator gets overloaded by alarms in a way that she can not cope with the number of alarms produced. In this case, the IDS is of no help for the network's operators.

By taking into account easily accessible context information about the hosts in the network, we will show in the later sections of this work, that it is possible to reduce the amount of false positive alarms for more than 95%. The information includes the hosts' operating system and the applications running, together with their version numbers.

In order to evaluate the effectiveness, we build a prototype implementation (called "XYN") as a meta system, which re-samples the output of an IDS system. We have chosen Snort as a prominent example of an IDS due to its well defined interfaces. The prototype was developed and tested in cooperation with a German ISP for high quality customers and businesses, which gratuitously setup the IDS in the DMZ of the network.

Our contribution to this area is twofold:

- We discuss technical and organizational means to collect and structure the additional information on the context regarding the hosts in the network.
- We present an architecture that works on top of an off-the-shelf IDS and demonstrates the feasibility of this approach.

2 Related Works

In [10] an approach is discussed, with the purpose to reduce the number of positive alerts by clustering single events which belong to the same attack context. Even though the number of generated alerts is reduced, it does no significant effort of removing invalid alerts. To a certain extend, their work can also be used to reduce the number of false negative samples, which is not covered in our paper. Thus both systems could benefit from a mutual integration, but are independent otherwise.

Another approach to correlate data from multiple sources together with primary data from an IDS is developed in [1]. However the authors do not consider databases which include context information, like e.g. the configuration and applications running on the target systems. Unfortunately the actual correlation process it is also a bit unspecific, i.e. there is no description on how the reduction of false positive and even false negative errors is achieved.

The Basic Analysis and Security Engine (BASE)[2] is a frontend for end-users and administrators. The application accesses directly the database of an IDS, which is extended with additional

[2]http://base.secureideas.net/

tables. The objective of BASE is to aggregate and re-sample frequent alarms and store the report back into a new format. The newly created records set can then be analyzed under different aspects, e.g. specific time, signature of attacks, IP-addresses, etc. However BASE does not make an additional qualitative check on the data, therefore it is no help in distinguishing between true and false positive alarms.

Scott proposes a similar concept in [9]. His motivation was the raising burden for system operators who are working on growing output of network based intrusion detection systems. The lack of a unified format for all the data which is collected from different firewalls, routers, and IDS, poses as a significant hurdle to them because there are no standardized interfaces. Before the data can be used as input for algorithmic processing, it has to be stored in databases. At this point a system should have knowledge about the desired security level of the machines present in the network, applications and services in the respective subnetworks, as well as allowed interactions between the machines. With the help of this aggregated information an evaluation could be initiated to decided upon an alert to the administrator.

In fact, the architecture of Scott's work is conceptual similar to our work, but there are two decisive distinctions: first, his work is more complex than ours, integrating routers and firewalls. Second, however, the proposed architecture of Scott is knowledge neither implemented nor tested with real data.

3 Technical Background

Intrusion detection systems can be divided into two different classes: host-based intrusion detection systems (HIDS) and network intrusion detection systems (NIDS). An host based IDS runs on a local machine. It knows the exact environment and can therefore recognises a system specific attack on that system. A network IDS monitors network traffic; either of the complete network, or there is one NIDS for each network segment. By analysing the traffic the NIDS can detect various attacks against one or more hosts in a network. Unfortunately a NIDS is less specific as a HIDS, since no information on the different hosts are available. This leads to serious problems: Today many vulnerabilities are tested automatically by exploited machines regardless whether a machine is exploitable or not. In most cases attacked machines are not vulnerable, since they do not fulfill the requirements for a tested exploit. The fact that a NIDS has no specific information on the different host leads to a huge amount of false positives. Therefore a NIDS becomes simply ineffective in an huge network.

A Common Intrusion Detection Framework was defined in [4]. With the help of this framework, even the interplay of IDS from different vendors was targeted. The framework is designed to be modular and consisted of several parts: one box for collecting data ("E-Box"), the actual unit for processing ("A-Box"), the storage unit ("D-Box"), and the part for sending alerts ("R-Box"). Data is transmitted between the boxes with the help of the *Common Intrusion Detection Specification Language* CISL. The E-Box's task is to collect data relevant for the intrusion detection. It is crucial to avoid that data are added, removed or changed by this unit. Problems at this stage include network packet fragmentation, timeouts, and stream reassembling. This stage also has to detect and compensate known methods of so called IDS-evasion [6]. The collected data is analysed in the A-Box. Analysis is usually done with the help of either anomaly detection, or based on matching signatures (sometimes also called "misuse detection") [3]. The first approach makes use of statistical methods, data mining or similar, to distinguish between different types of traffic and

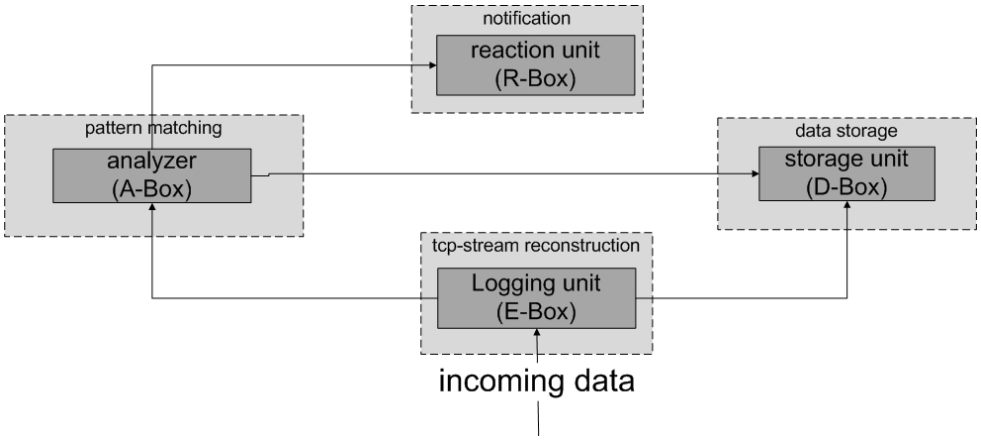


Figure 1: CIDF-Design

detect unwanted behaviour. The other method uses an extensive list of signatures which describes irregular data packets. However, neither approach has been up to now been proven to be superior as both have their type of failures. Due to the fact that our approach needs detailed information of the attack, we work with a signature based IDS in this work.

Therefore, we have chosen Snort [7] as a NIDS which is based on the CIDF architecture. Snort only differs from the CDIF architecture in the fact that the D-Box is merged together with the R-Box. The main task of Snort in our setup is to collect and save recognised attacks as events in a MySQL database. An important feature of Snort is that it often saves an event together with a CVE (Common Vulnerabilities and Exposures) reference. Such a reference is named by the Code 'CVE' , the year it appeared for the first time and a continuous number. A valid reference name would be CVE-2003-0034. Due to the global unique naming one can find detailed information about affected software on various sites within the international security network. Standardized information in XML-format can be found e.g. in the National Vulnerability Database (NVD)[3].

3.1 Type I and II errors

As we shortly elaborated in the introduction, it is inevitable for every IDS to make false positive and false negative decisions. In our approach we apply context based decisions on all alarms, i.e. true positive and false positive alerts. This is depicted in the following table:

Original decision	Intermediate result	Decision of XYN	Final result
True positive	Correct	Accept	Correct
		Reject	*Type II error*
False positive	Type I error	Accept	Type I error
		Reject	*Correct*
True negative	Correct	no reaction	Correct
False negative	Type II error	no reaction	Type II error

[3] `http://nvd.nist.gov/`

As can be seen from the list above, our approach raises the number of type II errors (first italic phrase), while reducing the number of false positives (second italic phase). We will discuss the extend to which we were able to reduce errors of type I, and in turn raised the number of errors of type II, in Section 5.

4 Prototype

In this Section we will describe the conceptual architecture of our prototype XYN and give details of the implementation.

The architecture of XYN can be divided in four different functional units: the first unit collects the events found by Snort. The second unit imports and extracts detailed information about a vulnerability from a vulnerability database, i.e. the National Vulnerability Database. Due to the description of a vulnerability, XYN is able to extract information on the affected operating system as well as the affected software. The third and fourth part are described in the two upcoming Sections in more detail.

4.1 Collecting Context Data

The third part of XYN is responsible for collecting data about the different hosts in the network. We will now describe two different methods, to collect this data – it should be noted that they can be combined, if necessary.

If the IDS is set to protect the network of a company, there are good chances that the IT is managed in accordance to the *Information Technology Infrastructure Library* (ITIL)[4]. One part of the ITIL-definitions is the *Configuration Management Database* (CMDB). This database should contain the configuration parameters of all hosts, including the complete history, with details about hardware, software, processes, infrastructures, responsibilities, and other components. Depending on the critical level of the respective hosts, the level of detail is either very high, or rather low. For critical infrastructures like routers, switches and central servers, the CMDB should include in any case the operating system, its version, as well as all applications together with their versions and patch levels.

Using such a database, it is trivial to derive the necessary information which is needed for an IDS.

In the case of smaller networks, there is usually no CMDB available. Then, one out of several tools can be used to build up a basic database of the required information. As these approaches are to some extend prone to errors, the results should in any case be cross-checked by some human system operator.

One known tool for this is Nessus [8], a network vulnerability scanner. Nessus starts with a portscan in order to detect open TCP and UDP ports. For each of these ports, a number of plugins are used in order to determine the application and possible security issues. The results are then written into a XML-file which can be parsed and transfered into the XYN database.

Another tool for this purpose is `nmap` [5]. It has a similar feature range as Nessus, but misses vulnerability scanning. However, nmap has one of the best methods to detect aribtrary applications running on open ports, even if the ports are non-standard. To this end, nmap makes use of so called

[4] http://www.itil.org/

"banner scanning", where typical network protocols are send to each open port, until a reasonable reply is received for one of them.

Further refinement of the results provided by Nessus or nmap can be done with behavioral scanners. These are specialized tools which send a variety of specific requests to a server and compare the answers of the server to a database of known implementations. With the help of these scanners it is even possible to correctly identify implementations of network servers which try to hide their identity by spoofing their banners.

Finally, if one port features an HTTP-server, there might a multiplicity of scripts and applications running on a single TCP port. In this case we recommend the use of a crawler to determine active applications.

We conclude that there is a multiplicity of ways to collect the appropriate set of necessary information.

4.2 Deciding on Relevance

The final part of XYN decides if an event is a relevant attack against a host. This decision is based on the information given by the other units. If the unit recognises this event as an attack, then a notification will be send to the administrators. Otherwise this event will be categorised as irrelevant. This is true if there is at least one requirement of the attack which is not fulfilled by the target system. For instance, if the software on the target system is a version, which includes a patch against this attack.

Translation from event information to affected software/OS is made via CVE references. As described above those references guarantees global unique naming of known attacks.

One problem is that not every event recognises by Snort includes a reference to a vulnerability. On the one hand most of these events are only harmless port scans. On the other hand those events cannot be simply dropped since this could have the unwanted effect of increasing false negative rate.

5 Evaluation

For the evaluation of the success of our approach, we were able to place the prototype in a DMZ of a German ISP. The events observed by the NIDS Snort were stored in a MySQL-database running on a Debian machine.

In this DMZ there are more than 60 hosts to be monitored by a NIDS over periods of seven days. Every packet passing the DMZ was therefore send to the NIDS.

In order to verify the results we did several runs of one week each, the resulting numbers were reproducible and had significance.

5.1 Frequency of signature

In total 244 different signatures triggered 771,733 alarms. Figure 2 lists all signatures which where triggered 40 times or more often together with the fact if Snort provides a CVE-reference to them, or not. As discussed before, our approach bases on the presence of CVE-references in order to decide upon an attacks' relevance.

However, only 65% of all attacks were triggered by an signature containing a CVE-reference. On the other hand, a huge number of alarms where of type `portscan` or `http_inspect`. Both of these to not refer to an actual attack, but rather to an suspicious data packet which, by itself, is of

occured events	signature-name	signature-class	cve-reference
49	(http_inspect) IIS UNICODE CODEPOINT ENCODING		
52	WEB-PHP remote include path	web-application-attack	
92	WEB-PHP admin.php access	attempted-recon	X
94	WEB-MISC nessus 2.x 404 probe	attempted-recon	
103	WEB-CGI w3-msql access	attempted-recon	X
107	WEB-CGI awstats.pl configdir command execution attempt	attempted-user	X
153	(http_inspect) WEBROOT DIRECTORY TRAVERSAL		
177	(http_inspect) DOUBLE DECODING ATTACK		
189	DOS utf8 filename transfer attempt	suspicious-filename-detect	X
207	(portscan) TCP Portsweep		
209	SNMP MS Windows getbulk request	attempted-admin	X
231	WEB-MISC cross site scripting attempt	web-application-attack	
233	WEB-MISC encoded cross site scripting attempt	web-application-attack	
278	WEB-IIS cmd.exe access	web-application-attack	
354	ICMP Destination Unreachable Communication Administratively Prohibited	misc-activity	
379	WEB-MISC /etc/passwd	attempted-recon	
424	WEB-MISC robots.txt access	aplication-activity	
535	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY		
775	WEB-MISC cgitest.exe access	web-application-activity	
840	TFTP Put	bad-unknown	X
1546	DNS SPOOF query response with TTL of 1 min. and no authority	bad-unknown	
3266	WEB-MISC SSLv2 openssl get shared ciphers overflow attempt	attempted-admin	X
4470	WEB-CGI redirect access	attempted-recon	X
84433	(http_inspect) BARE BYTE UNICODE ENCODING		
178836	(portscan) Open Port		
494980	SMTP ClamAV recipient command injection attempt	attempted-admin	X

Figure 2: Frequency of Signatures

no danger for a modern system. If we removed these packets from our analysis, about 1% of alarms remained without CVE. Thus, ≈99% of all alarms are either no attack, or posses a reference where further information is available.

5.2 Attacked Software

Figure 3 illustrates the targeted software by the exploits and the software running in that network. According to privacy reasons, the hostnames have been replaced with pseudonyms. Whenever the targeted software version matches the running software, this is marked in the last column of the table. In this case however, we did not check for the software version, but only for general relevance.

5.3 Overall Analysis

In this Section, we will evaluate the effectiveness of the deployed prototype. The evaluation's success obviously depends on the deployment scenario – thus it is very likely that in a different setup, i.e. at a bank, within a governmental agency, or at some completely other place, the prototype will perform differently. As we tested our setup in the DMZ of a medium-to-large German ISP we are confident that the attacks are representative to a large degree for all major networks.

Within one observation time frame of one week, our framework noticed 771,733 attacks, out of which 567,421 were related to a CVE-reference. If we subtract from the remaining number of alarms those which only were of informational nature and did not refer to a real attack, e.g. warning on portscans and `http inspections`, only 1% of all attacks where without CVE-reference. These alarms had to be send directly to a human administrator for manual inspection. These were a total of 8567 alarms. Even if this number appears to be high at the first glance, this number can be further reduced by aggregating the alarms and sending them out in discrete intervals only. As in our setup the post-processing took place once per hour. There were only a total of 385 messages delivered to a human administrator this way.

occured events	hostname	application	vendor	relevant
40	trial1.host.net	Windows XP	Microsoft	
40	trial1.host.net	Windows 2000	Microsoft	X
40	trial1.host.net	IIS	Microsoft	X
40	trial1.host.net	FrontPage Server Extensions	Microsoft	
40	trial2.host.net	AWStats	AWStats	
40	trial3.host.net	AWStats	AWStats	
42	trial1.host.net	IIS	Microsoft	X
42	trial1.host.net	Apache	Apache Software Foundation	
43	trial4.host.net	Windows Server 2003	Microsoft	
46	trial5.host.net	NetMeeting	Microsoft	
48	trial1.host.net	Sambar Server	Sambar	
48	trial1.host.net	Savant WebServer	Michael Lamont	
52	trial5.host.net	Windows ME	Microsoft	
52	trial5.host.net	Windows NT	Microsoft	
52	trial5.host.net	Windows Server 2003	Microsoft	
52	trial5.host.net	Windows XP	Microsoft	
63	trial1.host.net	mSQL	Hughes	
78	trial6.host.net	Kernel	Linux	X
78	trial6.host.net	tftp	tftp	
92	trial1.host.net	PHP-Nuke	Francisco Burzi	
95	trial7.host.net	Mailman	GNU	
153	trial5.host.net	OpenSSL	OpenSSL Project	
1550	trial1.host.net	ClusterCATS	Allaire	
4418	trial8.host.net	ClamAV	Clam Anti-Virus	

Figure 3: Attacked Software

From the alarms which had a related CVE-reference, we found that 95.4% of them could be discarded, if the software and the software version on the target system were known and accurate. Even if the version of the software at the target system would have been unknown, 94.3% of these alarms could still have been discarded. These numbers were found as averages in several runs, where the variance was generally found to be well within less than 5 per cent points.

Summarizing the results, our framework was able to reduce the number of alarms on 737,007 alarms, which is about 95%.

In all runs we did not notice that an alarm was removed by the system (see Section 3.1). We considered that this is due to the nearly perfect information about the hosts to be protected due to the use of a CMDB. In addition, the overall quality of information gathered from CVE-references can be considered very high. Thus we conclude that, while it is of course possible that our approach might eventually remove relevant alarms from the notice of administrators, the probability is rather negligible compared to the huge number of false positive alarms removed.[5]

Even as we were unable to make an in-depth analysis of the remaining $4-5\%$ of attacks, we can still conclude that the number of false positive alarms was reduced at least 95.5%[6].

6 Discussion and Conclusion

In this paper we have introduced a prototype of an IDS which strongly reduces the amount of false positive alarms. Instead of creating a new IDS from scratch, our prototype works with the alarms

[5]Please note that we do not claim that our approach will never result in an increase of false negatives. We want to express that the probability is very small.

[6]The real number is within the interval of 95.5% to 100%.

generated by a NIDS. All of these alarms are checked for relevance before being actually send to a human administrator.

To this end, XYN was designed to be highly modular, where the information is pre-processed, processed and stored by different modules. The only task of the central core is to correlate and evaluate the basic information provided.

The amount of reduced alarms highly depends on the quality of the available information about the hosts in the network. Especially critical are all cases, where the impression exists that no attack actually took place:

- if the NIDS has not recognized the original attack, it is in our model impossible to correct this error.
- if the NIDS uses an error-prone database for its signatures it can happen that an attack has not proper information on the vulnerable targets. This can lead to a wrong de-selection of the respective attack. This typ of error can also appear, if there is no signature for a given attack.
- if the database of XYN misses a vulnerable application.

To avoid the cases listed above, only data with a known high quality should be used. As the first and second case describe errors which happen in the NIDS, our approach has no direct influence on them, other than choosing a well supported NIDS. Case three was to a certain extend discussed in Section 4.1. Obviously, the best case is to adhere to ITIL-conform IT-management, where the third case should only happen within a negligible frequency.

The results achieved with this solution show that it is feasible to vastly reduce the amount of false positive alarms. In addition to that there are even more possibilities for future work and enhancements:

- combination and automation of collecting context information for the database containing the list of vulnerable applications.
- finding possibilities to exclude certain applications from being relevant for some attack signature, even if no CVE entry is available.
- optimizing the frequency of the interval in which the central core module is run. A balance has to be found between early notification and the number of alarms.

6.1 Conclusion

In this paper we have showed that, using appropriate context information, we can reduce the number of false positive errors in an NIDS. While it is difficult to extrapolate the results from our setup into different scenarios, we are still confident that the reduction of more than 95% of false positive alarms can be reproduced within other networks. The cost for achieving this enhancement is rather small, if an CMDB is already available – in other cases we think that the initial effort will pay-off with the enhanced functionality of the NIDS.

References

[1] J. Afonso, E. Monteiro, and V. Costa. Development of an Integrated Solution for Intrusion Detection: A Model Based on Data Correlation. In *ICNS*, page 37. IEEE Computer Society, 2006.

[2] J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, Fort Washington, Pennsylvania, USA, April 1980.

[3] S. Axelson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Chalmers University of Technology Department of Computer Engineering, Göteborg, Sweden, March 2000.

[4] C. Kahn, P. A. Porras, S. Staniford-Chen, and B. Tung. A Common Intrusion Detection Framework. Journal of Computer Security, 1998.

[5] G. Lyon. Nmap, a free and open source utility for network exploration or security auditing. `http://nmap.org/`. last visited September 2008.

[6] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[7] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *LISA*, pages 229–238. USENIX, 1999.

[8] T. N. Security. Nessus, vulnerability scanning software. `http://www.nessus.org/`. last visited September 2008.

[9] Steven J. Scott. Threat Management - The State of Intrusion Detection. http://www.snort.org/docs/threatmanagement.pdf, August 2002.

[10] J. Zhou, M. Heckman, B. Reynolds, A. Carlson, and M. Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

Investigating the Benefits of Using Multiple Intrusion-Detection Sensors

Magnus Almgren Erland Jonsson
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Magnus.Almgren@chalmers.se
Erland.Jonsson@chalmers.se

Abstract

Most intrusion detection systems (IDSs) available today are using a single audit source for detection, even though attacks have distinct manifestations in different parts of the system. Previously, we have explored the benefits of combining several sensors monitoring different audit sources to improve the detection of attacks. In this paper we go one step further and investigate possible synergetic effects by actively sharing information *between* distinct intrusion detection sensors taking events from isolated audit sources. We present four scenarios where we show how the function of one IDS, measured as false alarm rate, performance in terms of used resources, or attack response, can be improved by having access to information collected and analyzed by another IDS. Based on these four scenarios, we then generalize our findings and outline necessary properties of a sensor communication framework for multiple IDSs. Our focus is on cooperation between IDSs, but we also touch on response techniques.

Keywords: intrusion detection, IDS cooperation, IDS response

1 Introduction

Other studies have emphasized the advantages with using several complementary intrusion detection systems (IDSs) to improve the attack detection coverage as compared to a single system ([3],[20]). One can mix different analysis methods, such as the system proposed by Tombini et al. [20], where they serialize the analysis by first using an anomaly detection engine followed by a misuse detection engine. One can also collect different security-relevant events from a variety of sources and compare the resulting analysis by the corresponding IDSs, as done by Almgren et al. [3]. For many of the suggested complementary IDS deployment schemes, and the ones above in particular, the IDSs in question are relatively unaware of the other pieces of the system. In this paper, we explore the synergies that can be harbored from using different systems, given that *these systems know of each other's existence and can exchange different types of information.* By actively cooperating, we hypothesize that one may gain better performance in terms of used resources, improve the attack detection coverage, and possibly limit the risk of questionable behavior through better response.

Our work is empirically built around four scenarios. In each scenario we present a problem, followed by a description on how information available to one intrusion detection sensor would be of use to another sensor, sometimes directly and other times indirectly through a better understanding of the resulting alerts. The advantages does not always come without a penalty, so we also include possible disadvantages with the approach. We have also built a proof-of-concept in our test bed for these scenarios. Contrary to the prevalent approach of describing the experiments in a separate section, we interweave a discussion of the implementation directly after the scenario description. This makes the presentation

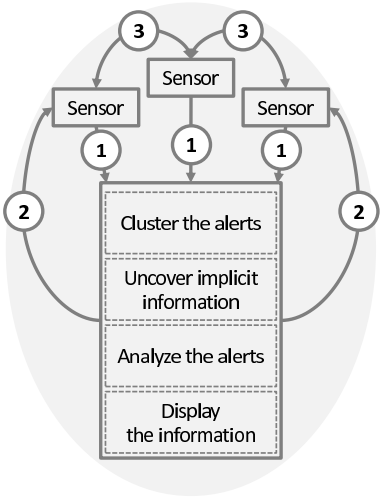


Figure 1: Model of a Sensor Communication Framework.

more cohesive, and further let us save space as the scenario description does not need to be repeated. Based on the experience implementing the scenarios, we propose an outline for a sensor communication framework. This framework is briefly presented *before* the scenarios, even though it is the result based on the analysis of the scenarios. Introducing the outline before the scenarios allows us to refer to the framework throughout the paper. By the end of the paper, we return to the framework and describe it in more detail.

The aim of the paper is to explore the benefits and disadvantages with cooperating intrusion detection sensors, and identify the building blocks necessary for using diverse sensors efficiently. Thus, we see our proposed outline for sensor communication as a guide for future research and development and not as a finished proposal.

The rest of the paper is organized as follows. In Section 2 we outline the sensor communication framework. In Section 3 we describe the four scenarios that form the basis for the paper, including their implementation. We then consider the result in Section 4 and return to the sensor communication framework in Section 5. We go through related work in Section 6. We describe future work in Section 7 and the paper is concluded in Section 8.

2 Outline of the Sensor Communication Framework

In Figure 1, we show an outline of a sensor communication framework. There are three modes of communication and these are marked with a number. The first mode, sensor-to-console communication (marked with 1 in the figure), is a necessary piece to any IDS as the site security officer (SSO) needs to be able to analyze all alerts. Most sensors have different output options, even though they in some cases are quite rudimentary. We are going to assume that all sensors are able to forward their alerts to a console. For more advanced sensors, there is also console-to-sensor communication (marked with 2). Finally, there may also be sensor-to-sensor communication (marked with 3).

We also show some of the components of an *alert-decision framework* (the rectangle) that we argue is crucial when collecting alerts from several sensors. It has not been implemented for this paper, but its parts have been identified. In the scenarios below, we refer to Figure 1, and then expand on its parts in Section 5.

3 Scenario Descriptions

In this section we describe the scenarios that are the basis for the paper. For each scenario, we start with describing a problem and then outline how the problem can be solved, or at least mitigated, by combining several IDSs. We concentrate on three audit streams and the corresponding sensors that analyze the information found in these streams. For simplicity, we use the term "sensor" and "intrusion detection system" interchangeably.[1] We choose to look at a network-based IDS (with the network as the audit stream), an application-based IDS (with events collected from within the application or from its log file), and finally a host-based IDS (HIDS) analyzing system call sequences. We need to limit the scope of the paper, and we choose this combination as there exist mature systems for us to use (Snort, Apache), or we have previous experience with them (webIDS). We do not consider this as a restriction of our work, as other sensor / audit sources would most likely also work in a similar fashion (see for example Scenario 4). Further details of the experiments are found in Section 3.1.

Each audit stream contains a subset of all security-relevant events that can be collected. When comparing two audit streams, some events directly overlap and exist in both streams. In other cases, the information exists in both streams but it can only be compared after a reconstruction step by the IDS. For example, a network-based IDS that preprocesses the collected events from the network can rebuild the HTTP protocol and the requests to the web server as seen by an application-based IDS. This reconstruction may be time consuming and also be circumvented by a wily attacker [18]. However, there also exist events in an audit source that cannot in practice be reconstructed from the information found in another audit source. A typical example is that a NIDS cannot predict the system calls invoked by a cgi program resulting from a request to the web server.

In the rest of this section, we describe a number of scenarios where the exchange of information is useful. These range from sharing information that are more easily collected from one audit source, but helpful when analyzing the information found in another audit source, to changing the security policy in one sensor based on input from another sensor.

3.1 Details of the Experiments

Even though we have implemented a proof-of-concept for the examples, we focus on the concepts described in each scenario and not on the particulars of the implementation. We are also limited in space, and for these reasons the discussion of the implementation and the resulting tests are quite terse. However, there are lessons learned from the implementation and we try to highlight these for each scenario.

As mentioned before, we concentrate on three sensor / audit sources. We use Snort [19] as the network-based IDS. It is well-known, open source, and a de facto standard. As we have previous experience with a system similar to the *webIDS* ([2], [4]), we use the Apache HTTP Server [6] as a web server combined with an application-based IDS (e.g., the webIDS). As Apache has extensive facilities for adding functionality in the form of modules, as well as mature logging functions, the webIDS used for this paper is in reality a mixture between external code and directives within the web server configuration files. For simplicity, we consider both changes to the Apache configuration files and the external code to be part of the webIDS in the descriptions. Finally, we developed a very simple HIDS that monitor system call activity for cgi programs.

At this stage, the implementation of the scenarios is not mature enough to be run on a production server. As we wanted a proof-of-concept to validate our reasoning, we sometimes allowed a simpler solution even though it led to a performance degradation. One would most likely prioritize performance over simplicity for production servers.

[1] In complex deployments, an intrusion detection system may have several sensors but in a typical small-scale deployment, an IDS usually only use one sensor.

3.2 Scenario 1: Understanding the missing alert

Sensors: NIDS, webIDS
Exchange: "encryption status" of web requests

Description

As the first example, we consider a system with a network-based IDS (NIDS) and an application-based IDS (webIDS) monitoring a web server. Using an attack that both the NIDS and the webIDS can discover, we expect that we either have no alerts (meaning no attack is in progress) or alerts from both systems simultaneously. Unfortunately, as the sensors are imperfect there are also situations when there is an alert from only one sensor requiring a more careful analysis on the part of the site security officer (SSO). For example, if there is only an alert from the webIDS we have the following two possibilities.

- The webIDS failed its attack analysis and wrongly reported an attack (false positive), or
- the NIDS failed to discover the attack (false negative).

Distinguishing between these two situations is rather tricky, especially given the real-time nature of intrusion detection. If a real attack is ongoing, the SSO needs to block it but if it is a false alarm the alert can be ignored. Any indication to which scenario is the most likely helps the SSO in his analysis. Unfortunately, most intrusion detection systems today do not give an indication of the certainty of their predictions. However, this is exactly the aspect we explore in this scenario. By considering the failure modes of a sensor, we know when its analysis is unsound and thus when its alerts should be marked with a very low probability of being true.

For this example we concentrate on a particular weakness of a network-based sensor searching for web attacks; it is blind to attacks sent over an encrypted channel. The NIDS can then no longer reliably analyze the request and any attempt of deep-packet inspection is wasting computational resources. Thus, given that the request is encrypted the NIDS should mark its analysis as *incomplete*. Unfortunately, the NIDS cannot easily by itself report whether a request is encrypted, because, as the channel is encrypted, the NIDS does not even know of the request in the first place.

The webIDS, on the other hand, does not suffer from the same weakness to encrypted content. From the perspective of the webIDS, an encrypted request is not much different from any other request. The fact that the request is sent over an encrypted link is directly available within the event stream, but so is the request in clear text. The webIDS can thus analyze the request regardless of its encryption status.

In this case, the webIDS has direct access to the encryption status, i.e., a piece of information that is critical for explaining the ability of the NIDS to detect attacks, even though it is of little value to the webIDS itself. Sharing the information with the NIDS may avoid time-consuming deep-packet inspection, and including this fact in an alert explains *why* an expected alert from the NIDS is missing for the SSO. Knowing that the HTTP request was encrypted in the example above thus directly invalidates the NIDS contribution of "no attack," and the SSO can start with assuming there is an ongoing attack until further evidence is collected.

To generalize, there exists information that is much more accurately collected with less performance overhead from one audit source compared with another. However, this information may still be of value to other IDSs that cannot directly tap the original audit source in question. The information does not even have to be analyzed and processed before being shared, as in this example.[2]

Experiment

We deployed Snort and the webIDS together with the Apache web server in our test bed. We added an extra signature to the webIDS, so that it would alert for all requests that were sent over an encrypted

[2] In Almgren et al. [5], we explore this scenario further.

channel. All alerts were forwarded to a central console, thus only using communication mode 1 from Figure 1.

We tested the system by using the `phf` attack [10]. We first ran the attack without encryption, and this resulted in a Snort alert and a webIDS alert. We repeated the experiment over an encrypted channel, resulting in a phf alert from the webIDS but no alert from Snort. However, this time we also had an alert from the webIDS indicating that the request was sent over an encrypted channel.

By considering the alerts from Snort and the webIDS in conjunction, the SSO has a better picture of what is happening in his network. The extra webIDS signature for when the request is sent over an encrypted channel was quite easy to add. One can instrument the Apache web server to log the encryption status of a request directly, and, if the webIDS is already using the log file as its audit source, only a few changes are necessary to also incorporate this new signature.

However, there are still a few problems and these reflect the parts we have added to the alert-decision framework in Figure 1 (the rectangle). First, the alerts at the console should be correlated and merged into clusters if they are related. In our rather simplified experiment there was no problem for the SSO to manually cluster the alerts that belonged together. On a production server with a higher traffic load, there would have been many more alerts resulting in the manual analysis being quite laborious. Furthermore, we found that it is rather difficult to see that a piece of evidence is *missing*, i.e., that the alert from the NIDS is not there when the request is encrypted. Having the system automatically highlight *expected information*, such as a missing alert due to encryption, would have made the analysis simpler for the SSO.

3.3 Scenario 2: Only use the blunt tools when necessary

Sensors: NIDS, webIDS
Exchange: access of “sensitive resources”

Description

One of the major problems with intrusion detection systems are their propensity for false alarms. Some rules are not used for real-time monitoring within production environments because they cause too many false positives, even though these rules make it possible to detect attacks that would otherwise be missed. The cost of handling the false alarms outweighs the utility for these rules. Simply put, it is difficult to write good rules that alerts for most attack variants, do not alert for normal behavior, and is simple enough to be used for real-time analysis.[3]

In this scenario, we consider how to profit from the detection capability of such rules, but without suffering from an excessive number of false alerts. We use the Snort rules that scan traffic from the server to the client for *sensitive content* as an example. These rules search for strange or invalid output from known programs, e.g., if the `id` program is run and returns `root` (SID 498). The Snort rule with SID 498 is described at [7], including a discussion of possible false positives. Such rules are thus similar to the bottleneck verification described by Cunningham et al. [8]. In short, if an attacker runs any of a set of programs on the host to verify his status, the attack attempt is detected regardless of the original attack vector. Even though the attacker changes the attack vector, the output from these programs is more difficult to change, thus giving a last recourse to detect a successful attack.

To limit the number of false alarms, we use these rules *only when* there is a significant risk of an attack taking place. For example, when a resource that is deemed as relatively safe is accessed, such as a static web page, these rules are turned off. When a sensitive resource that may be hacked is accessed, e.g., a cgi resource with a known weakness, these rules are turned on. Hence, we avoid all the false alarms caused by normal behavior when accessing the majority of the safe resources.

[3] See for example Julisch [11] for a detailed analysis of false alarms.

Even though a network-based IDS would be able to guess when a sensitive resource is being accessed (cgi-bin in the path name), its analysis is relatively inexact. The webIDS, on the other hand, can easily determine the exact mapping between URIs and file resources done by the web server. If a sensitive resource is accessed, the webIDS can adapt its own analysis and also share the information with other components so that also their analysis is more detailed. Knowing that a *sensitive resource is being accessed* should be of interest to most components. All sensors can then use more rules to analyze the traffic, or, in the case of an anomaly-based system, lower their threshold value for normal traffic.

Experiment

We let the webIDS collect information about what type of resource is being accessed in response to the web request. If this resource is sensitive (here taken to be any cgi program), this fact is communicated to Snort. Snort can then also analyze the traffic going back to the client for strange patterns.

We added an extra HTTP header to the response from the web server, reflecting the type of resource that created the response. In this simple example, we could directly add this information within Apache but more complex schemes would have required a more elaborate instrumenting of the webIDS. We then created a new rule that directs Snort to search for the extra header as well as the original attack pattern found in the Snort rule with SID 498 [7]. Thus, we used an indirect variant of communication mode 3 shown in Figure 1. The webIDS added information that Snort would intercept and interpret, even though it did not directly setup an explicit communication channel to Snort.

To test the result, we created one static page containing the output from the `id` program, as well as a dynamic resource (e.g., cgi program) that returned a page containing the very same output. When we requested the dynamic program, both the original rule and the new modified rule raised an alert in Snort. However, when we asked for the static page, only the original rule raised an alert. Thus, the new Snort rule only raises an alert when a sensitive resource created the response.

An alternative approach is the following. Instead of the the direct sensor communication (mode 3), we only let the sensors communicate with the console (mode 1 in Figure 1). We let the webIDS raise an alert when a sensitive resource is being used and leave the Snort rule unchanged. Alerts from both Snort and the webIDS is sent to the console, where the alerts are correlated. By adding a rule to the alert analysis module (found in the rectangle in Figure 1), an alert cluster is only forwarded to the SSO if it contains both an alert from the webIDS and from Snort. We did not implement this approach, but it shows that we can achieve the same end result from the point of view of the SSO without having a channel between the webIDS and Snort (mode 3). In both approaches, the SSO only has an alert if a sensitive resource has been accessed and the returned response is the output from the `id` program.

3.4 Scenario 3: Only call for the heavy artillery when the performance penalty is affordable

Sensors: NIDS, webIDS, HIDS
Exchange: indication of “possible attacks”

Description

Frequently, an alert only gives one piece of the puzzle; it is only an indication and not sufficient evidence of a possible ongoing attack. As the IDS monitors a single audit source it does not have the complete picture of what is happening in the system, and can thus only give an indication that an attack may be in progress. For example, Snort alerts when only finding the string `phf` in a web request, regardless of whether the cgi resource `phf` exists on the web server, it being vulnerable to this particular attack, or what the final outcome of the attack is. Thus, the SSO needs to have a comprehensive background

knowledge of the details of his system and then carefully investigate each alert manually to find out why the IDS reported an attack.

In this scenario, we investigate how a cross-system signature decreases the number of uninteresting alerts, i.e., the cases where Snort reports about the phf attack but where the attack is insignificant or where it failed. We let the webIDS invoke an extra analysis step only if the request contains a known vulnerable resource. This extra analysis step invokes a host-based sensor (HIDS) that carefully monitors the execution of the request by recording all its system calls and analyzing them in detail. As the resulting analysis is more detailed, the SSO should be more certain of an attack taking place when receiving an alert both from Snort and the HIDS. However, there is a severe performance penalty with the extended analysis and that is why it is not normally performed on each request. The server would simply become too slow and unresponsive.

As one requirement of intrusion detection is to have it run in real time, one sometimes must forgo a slower but more advanced analysis method. However, by only using such analysis for the cases where it really matters, as done in this scenario, it may be possible to absorb the performance penalty of the more advanced method and still run in real time. Thus, we first use a casual analysis to find problematic behavior. By *indicating possible attacks*, other IDSs can then perform a more detailed analysis. Furthermore, given that the resulting alert from this comprehensive analysis is less likely to be a false positive as compared to the original Snort alert, one may also use it as a basis for an automatic response.

Experiment

We deployed Snort, the webIDS and a small HIDS monitor that is invoked by the webIDS when a vulnerable request (here the `phf` cgi program) is detected by the webIDS. Thus, we use a variant of communication mode 3 for this scenario. The webIDS makes sure the HIDS only analyzes the requests containing a possible attack.

We first asked for a normal resource. There was no Snort alert and no alert from the webIDS. The HIDS was also not invoked by the webIDS.

We then asked for `phf` [10] four times. In two of the cases we used the program with normal arguments, and in the other two cases we tried to exploit it by including attack code that could list the password file. We also shifted between a vulnerable `phf` program on the server and a patched program. In all four cases, both Snort and the webIDS raised alerts and the webIDS invoked the HIDS. The HIDS raised an alert for only one case; when the vulnerable phf program was installed and it was accessed with the attack code resulting in an abnormal number of `execve` system calls from the trace.

Using the HIDS to analyze the request roughly slowed down the response by three times as compared to omitting the HIDS analysis. Even though not explicitly measured in this experiment, we suspect the number of false positives is reduced from the HIDS as it is only deployed when a possible attack is in progress (see Scenario 2).

3.5 Scenario 4: Enforcing a stricter security policy after suspicious behavior

Sensors: NIDS, webIDS
Exchange: “suspicious clients”

Description

An application-based IDS complements a networked-based IDS quite well. Even though their detection capabilities overlap, the NIDS also see many lower-level attacks that are not visible to an application-based IDS. However, the NIDS response capabilities are lacking when compared to an appIDS. Depending on the deployment, the NIDS is either running in passive mode with almost no response capabilities or in inline mode where it can block certain packets. Given that many alerts are false, one would like

response options tailored to the belief of the attack being real. The application-based IDS, being part of the actual application, has a fine-grained control over the application logic and offers more response options than the NIDS.

In this scenario, we are going to combine the detection capability of the NIDS with the fine-grained control over the application offered by the appIDS. We let the NIDS detect attacks, and let the appIDS block the source of these attacks from accessing any sensitive resources. However, the attacker can still access general (safe) information. Thus, even if the alert from the NIDS is false the client can still reach a limited set of functions.

We share the *list of suspicious hosts* assembled by one IDS with others in the system. Based on this information, an IDS can increase its analysis of traffic from these sources (similarly to Scenario 2 and 3) or enforce a stricter security policy. This type of information would most likely be of value to all sensors.

Experiment

We deployed Snort and the webIDS on our test bed and collected Snort alerts in a file located on the web server. When the webIDS detects that a dynamic resource is about to be run, it controls whether the client has previously been engaged in any attacks recorded by Snort. If so, the access to the dynamic resource is denied and the client is sent a response explaining the loss of functionality. If there has been no significant attack activity recently, the dynamic resource is run and its response is returned to the client. Thus, we used communication mode 3 from Figure 1.

To test the setup, we defined a specific cgi-bin program as being sensitive in the webIDS. We also defined a list of Snort alerts that would be serious enough to use as a basis for refusing access to sensitive resources. For simplicity, we let this be all Snort alerts related to the `phf` attack, as such attacks are easy to launch on our test bed. For this experiment, we used a naïve security policy and restored all access to a client after a predefined timeout. More complex security policies can also be implemented.

After the setup and the deployment, we tried to access the sensitive resource from a particular client. It worked. We then launched the phf attack from this client. After the attack registered with Snort, we tried to access the sensitive resource again. This time the web server did not access the sensitive resource. Instead, a response explaining why the resource was unavailable was returned. We waited for the predefined timeout in the webIDS and then accessed the sensitive resource again. As enough time had passed since the attack, the web server allowed access to the resource again by this client.

By a small tweak to this scenario, we would be able to replace the NIDS with a firewall that can detect general scans. Thus, the scenario would also be applicable to another sensor / audit source pair. Furthermore, by letting the firewall know about the webIDS, the firewall could adapt its own behavior. Access to any server lacking an appIDS would be blocked while access to servers with an appIDS would be allowed, as such servers would be able to provide a more suitable response compared to the options available to the firewall.

3.6 Summary

In Scenario 1, we eliminate some false negatives by considering the weakness of a NIDS to encrypted traffic. The fact that a request is encrypted is shared by the webIDS and *explains* why the corresponding alert from the NIDS is missing.

In Scenario 2, we share information whether *sensitive resources* are being accessed. If so, we use additional rules for the analysis. These rules cannot generally be used because of their false alarm rate.

In Scenario 3, the sensors share the results of their *attack analysis*, and, if there is cause for concern, an extra analysis step is performed. For general traffic, this extra analysis step cannot be run because it is too slow.

Finally, in Scenario 4, we share a list of *suspicious hosts* so that other units either can increase the sensitivity of their analysis or deploy a proper response.

4 Discussion

We wanted to explore whether a close cooperation between intrusion detection sensors would be useful, study the possible benefits and try to draw general conclusions from the practical examples. By using previous examples found in the IDS literature and brainstorming around these, it was easy to come up with the above four scenarios, chosen to focus on different advantages with information exchange between sensors. Building the proof-of-concepts were also quite simple. Apache is very easy to work with; it is modular and easy to extend. The logging facility in Apache is also very mature. Snort can also easily be extended, either in the form of new rules (easy) or through modules (difficult). We found it more difficult to change the modules, but this may just reflect the lack of experience we have with the inner workings of Snort. Let us consider the scenarios above, and what conclusions we can draw from them.

Observation 1 *Exchanging information between separate sensors has advantages.* In the four scenarios above we can see that sharing information is beneficial to the components in the system and there probably exist many other scenarios where this is true. We can improve the alert analysis for the SSO by including information from other sensors (Scenario 1). Both the false positive rate and the false negative rate can be improved (Scenario 2). Also the efficiency in terms of used resources can be improved (Scenario 3).

Observation 2 *Sharing raw audit information is seldom of use.* Customary, the analysis within an IDS is closely designed to match the type of events available in the audit stream the IDS monitors. For that reason, there are seldom cases where the raw audit information from, for example, a system call monitor is shipped to a network-based IDS. No scenario above shows this point, but it is rather the omission of such scenarios that corroborates this reflection. However, as some audit streams have similar events there are probably exceptions.

Observation 3 *Sharing refined information is useful.* As seen by most of the scenarios above, there has been some processing by the IDS before the information is of value to other sensors. From a practical point of view this is good news. As each IDS is quite self-sufficient and independent of others, there is no need to ship immense amounts of raw audit data across the system. One should instead consume and refine the audit data as close to the source as possible and only share specific results.

Observation 4 *Some collected information is probably valuable to all sensors in the system.* There is information that can be collected by any sensor and that is most likely of value to most other components in the system. In the scenarios above we touched on a few. When a *sensitive resource* is accessed, it is good if all sensors perform further analysis despite the possible performance hit. Similarly, sharing signs of detected *malicious behavior* or of *malicious sources* influence other sensors to increase their vigilance. Thus, we expect it to be easy to define a minimum set of information that each sensor should share with others in the system. However, we foresee that the practicalities of the sensor communication is difficult to achieve in practice.

Observation 5 *Some collected information is clearly only of value to a subset of sensors in the system.* Contrary to *Observation 4*, there is information that is only valuable for the analysis of the alerts from a single sensor. We showed such an example with the encrypted request in Scenario 1. Given that this application is so esoteric, it is unfortunately difficult for the developer of another sensor to foresee the need to share the information. For that reason, we find that it is very important to have an extensive logging facility as the one found in Apache.

Observation 6 *Exchanging information may lead to a performance overhead.* The flip side to sharing information, which we have already touched upon, is that we sometimes pay a performance over-

head when sharing the information. For example, consider Scenario 3 where we let the webIDS launch the HIDS when it detects the `phf` attack. Originally, we planned to let Snort detect the phf attack, send its alert to the webIDS, which in turn would launch the HIDS. However, this would have resulted in the web server waiting for the Snort attack analysis before handling each request. Thus, the details of the implementation is important and there may be a performance penalty to exchanging information.

Observation 7 *Using multiple steps in the analysis can improve its accuracy.* The final observation is not directly related to having IDSs exchanging information, but it is a direct observation from our implementation of the scenarios. We noticed that dividing the analysis into two distinct steps seemed useful. For example, in Scenario 2, we first detect whether a sensitive resource is run and, if so, we then use the special rules. In Scenario 3, we first detect an attack and, if so, we deploy the HIDS.

As can be seen in these observations, there are a number of advantages to the information exchange between the sensors. However, as already discussed, there are also disadvantages. First, exactly how should the communication happen between the components? This includes defining a format that is recognized by the different components (see *Observation 4*). A single vendor with multiple sensors can develop a propriety communication framework for his sensors. Open-source sensors can possibly be adapted by the community for inter-IDS communication. But as the IDS product field is divided between several actors, it is not easy to see how a general solution can be achieved. Second, as we listed in *Observation 6*, the communication also leads to problems for the sensors that have hard real-time requirements to keep up with the events in an audit stream. These sensors cannot wait for information from another part of the system before analyzing the current events. Third, another problem with the communication is that it creates a dependency between two sensors. What will then happen if one of the sensors is missing in a deployment scheme? If the NIDS depends on information from the appIDS for its analysis and this information is missing, the function of the NIDS may fail.

5 The Sensor Communication Framework

By returning to the outline shown in Figure 1, we now frame the discussion of the advantages by also considering the communication mode. Clearly, the disadvantages are closely tied to the practicalities of the communication between the sensors, so what advantages do we forsake if we limit their communication? As shown in Figure 1, we have considered three modes of communication and for each scenario we described the communication mode we used in our implementation. By summarizing the advantages, we distinguish three types of benefits. First, there are advantages to the analysis of the alerts. The SSO gains a better picture of what is happening in his network by combining information from several sensors. Second, there are certain performance gains by sharing information. Third, the attack response can be fine-grained, in that the component best suited for detecting an attack is not always the best suited for mitigating the consequences of this particular attack.

Can all the advantages be had by using the simplest communication mode (i.e., mode 1) in Figure 1? Unfortunately, we believe the last two advantages require communication between either the console and the sensors or directly between the sensors. In Scenario 3, the performance gain of deploying the HIDS *only when* another sensor detects an attack, implies a channel between the HIDS and the other sensor. Similarly, having the webIDS in Scenario 4 change its security policy based on information collected by another sensor also implies a channel between the sensors or from the console to the sensor. Thus, the performance gain can probably only be had if we solve the difficulties involving the communication outlined above. We would like to point out that this communication channel can also be indirect, as we showed in Scenario 2, where the webIDS injected extra information to the event stream for the NIDS to intercept and analyze (see also Lindqvist [13]). We expect such “tricks” make the implementation more efficient.

However, considering the first type, the advantages to alert analysis, we notice that we can achieve most of its benefits by only using communication mode 1 shown in Figure 1. This is the simplest communication mode and already common for the sensors as they need to ship their alerts to a central console for the SSO. In Scenario 1 and Scenario 2, we already explained how to use communication mode 1 to gain advantages to the alert analysis. By forsaking the performance gain by always running the HIDS, we can still achieve a good alert analysis in Scenario 3 using communication mode 1. All alerts would then be forwarded to a console and there be correlated.

Thus, the performance benefits described in Scenario 3 or the tailored response described in Scenario 4 require communication mode 2 or mode 3, but the advantages with several sensors for alert analysis can be had with only mode 1. However, using multiple sensors leads to more alerts for the SSO to analyze and, as we described in Scenario 1, the SSO would benefit from an alert analysis framework to automatically organize and present the alerts. The pieces of such a framework is summarized in Section 5.1.

This leads to two options for future research. Either one can explore a multisensor framework for better alert analysis (see Section 5.1), or one can further develop a common API between intrusion detection sensors and solve some of the communication hurdles to also increase the performance and response capabilities of the system as a whole.

5.1 Multisensor Alert Analysis

Using several sensors that forward their alerts to the SSO can easily make the alert analysis difficult. As we touched upon in Scenario 1 and Scenario 2, the SSO needs support in interpreting the alerts from several sensors. We identified four important areas based on our analysis of the scenarios. These four areas are shown in the rectangle in Figure 1.[4]

Alert Clustering First, related alerts need to be clustered, as explained in Scenario 1. Given that we use several sensors, the SSO will be overloaded with alerts unless we use a correlator (Kruegel et al. [12]). Even though this domain is the most developed of the four areas discussed here, we still feel that more work is necessary.

Adding Expected Information Second, realizing that also "no alert" carries information of value to the SSO, we need to add information even though it may be implicit. We gave the example of the missing alert in Scenario 1. Little work has been done to account for implicit information, but some tools have been developed (see for example Marty [14]).

Alert Analysis Third, different combinations of alerts in the cluster change the interpretation of the severity of the situation. For example, the fact that the web request was encrypted in Scenario 1 invalidated all Snort alerts. Given that the alert cluster does not contain the `sensitive resource` alert from the webIDS, the Snort alert with SID 498 is uninteresting and should be given a lower priority (Scenario 2). Automatic analysis of the agglomeration of alerts directly helps the SSO. Zhai et al. [24] present a framework for reasoning with complementary intrusion detection evidence, but further research is necessary to build an analysis engine that can handle contradictory evidence in the form of alerts.

Presentation of Alerts Finally, the alert cluster needs to be presented to the SSO. It should clearly indicate what information, including any implicit information, comes from what sensor, as well as an estimate of whether the alert cluster is relevant given the constituent alerts.

6 Related Work

With this paper, we wanted to explore whether a close cooperation between intrusion detection sensors is useful, study the possible benefits and try to draw general conclusions from several practical exam-

[4] General alert analysis also requires more postprocessing of the alerts as outlined by Porras et al. [15].

ples that we implemented. Even though other researchers have investigated properties of multisensor frameworks before us, we find that they have had different goals. Most presentations of multisensor frameworks concentrate on the attack detection by the different components. We have considered how any type of information available to one sensor could benefit another sensor, thus also including information not related to attacks (such as the encrypted request in Scenario 1).

Almgren et al. [3] investigate whether distinct sensors detect a series of attacks. They do not consider any cooperation between these sensors. Tombini et al. [20] combine an anomaly-based IDS with a misuse IDS. They discuss the attack detection advantages, but they do not in detail discuss other advantages or other sensor combinations. Other multisensor frameworks have also been described in the literature. Among these are EMERALD [16] and the STAT framework [22]. These frameworks provide a basis for sensor communication, and they also contain an alert correlator (i.e., *AlertStat*). However, the cooperative nature of the sensors are not emphasized, instead many of the components have been developed in isolation. HACQIT [17] and DIT [21] further explore the advantages of a multisensor framework by providing *intrusion tolerance*. They focus on tolerating attacks, and not, as we do in this paper, on presenting several advantages with cooperating intrusion detection sensors.

The most similar approach to ours is the *inquisitive sensor* described by Lindqvist [13]. He presents a number of scenarios where a more active sensor, i.e., one that not only forwards alerts to a console but also requests further information, would benefit the final alert analysis. Even though his work inspired us, he uses different scenarios from the ones presented in this paper and he lacks an implementation upon which to base a general discussion.

We have based the scenarios presented in this paper on ideas and common knowledge within the community. For example, Almgren et al. [2] listed the importance of keeping track of suspicious hosts almost a decade ago. Dagorn [9] explains the importance of sharing malicious requests between other sensors in her cooperative intrusion detection framework. Abad et al. [1] point out that some information about attacks will be distributed between different log sources, and combining the information should reduce false positives as well as provide stronger validation of an ongoing attack. Also others have discussed similar ideas, such as White and Pooch [23]. Our contribution is the consolidation and implementation of the examples, which form the basis for a more general discussion of the advantages of a multisensor framework.

7 Future Work

In this paper we identified several interesting problems regarding a multisensor framework. First, how can one extend current correlators to automatically include *missing* events in the cluster, based on the system configuration and the history of which alerts that previously were clustered together?

Second, how should a reasoning framework for alerts be constructed? A first outline of such a framework that can model the failure modes of an IDS as described in Scenario 1 has been completed. [5].

Third, current techniques for presenting alerts need to be extended so that they include the certainty of the alerts actually being relevant and also indicate possible missing information in an intuitive fashion.

8 Conclusions

We have described four scenarios where close IDS cooperation is advantageous from three different perspectives. It improves the alert analysis for the SSO, it improves the performance of the system as a whole, and it can sometimes lead to better attack response. Even though the analysis engine of a sensor is closely tailored to the types of security-relevant events found in the corresponding audit source, one can sometimes benefit from having limited access to information found in other audit streams.

Despite the clear advantages of a closer IDS cooperation, there are still many problems to be solved. First of all, there must be a standard for the information exchange. In some of our examples, we also suffer from communication delays, which affect the real-time analysis necessary for many sensors. Such penalties may be prohibitively expensive for production sites, unless their security requirements are extraordinary. However, it is possible to avoid some of these disadvantages by forwarding all alerts to an alert decision framework. We outlined the needed functions of such a framework.

Acknowledgments

The authors are grateful for valuable comments from our colleagues Ulf Lindqvist and Philippas Tsigas. This material is based upon work supported by the Swedish Emergency Management Agency.

References

[1] C. Abad, J. Taylor, C. Sengul, W. Yurcik, Y. Zhou, and K. Rowe. Log correlation for intrusion detection: A proof of concept. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 255, Washington, DC, USA, 2003. IEEE Computer Society.

[2] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In G. Tsudik and A. Rubin, editors, *Network and Distributed System Security Symposium (NDSS 2000)*, pages 157–170, San Diego, USA, Feb. 3–4, 2000. Internet Society.

[3] M. Almgren, E. Jonsson, and U. Lindqvist. A comparison of alternative audit sources for web server attack detection. In Ú. Erlingsson and A. Sabelfeld, editors, *12th Nordic Workshop on Secure IT Systems (NordSec 2007)*, pages 101–112, Reykjavík, Iceland, Oct. 11–12, 2007. Published by Reykjavík University, Iceland.

[4] M. Almgren and U. Lindqvist. Application-integrated data collection for security monitoring. In W. Lee, L. Mé, and A. Wespi, editors, *Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 22–36, Davis, California, Oct. 10–12, 2001. Springer-Verlag.

[5] M. Almgren, U. Lindqvist, and E. Jonsson. A multisensor model to improve automated attack detection. In R. Lippmann, E. Kirda, and A. Trachtenberg, editors, *Recent Advances in Intrusion Detection (RAID 2008)*, volume 5230 of *LNCS*, pages 291–310, Boston, MA, USA, Sept. 15–17, 2008. Springer-Verlag.

[6] The Apache Software Foundation. *The Apache HTTP Server Project*. `http://www.apache.org`.

[7] A. Chuvakin, N. Houghton, and A. Fischer. Details of the snort signature 1:498 attack-responses id check returned root. Internet. `http://www.snort.org/pub-bin/sigs.cgi?sid=498`. Accessed July, 2008.

[8] R. K. Cunningham, R. P. Lippman, and S. E. Webster. Detecting and displaying novel computer attacks with Macroscope. In *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, June 6–7 2000.

[9] N. Dagorn. Cooperative intrusion detection for web applications. In D. Pointcheval, Y. Mu, and K. Chen, editors, *The 5th International Conference on Cryptology and Network Security (CANS 2006)*, volume 4301/2006 of *Lecture Notes in Computer Science*, pages 286–302, Suzhou, Jiangsu, China, Dec. 8–10, 2006. Springer-Verlag.

[10] S. V. Hernan. 'phf' CGI script fails to guard against newline characters. CERT/CC; Internet, Jan 2001. `http://www.kb.cert.org/vuls/id/20276`.

[11] K. Julisch. *Using Root Cause Analysis to Handle Intrusion Detection Alarms*. PhD thesis, University of Dortmund, Germany, 2003.

[12] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion Detection and Correlation*, volume 14 of *Advances in Information Security*. Springer, 2005.

[13] U. Lindqvist. The inquisitive sensor: a tactical tool for system survivability. In *Supplement of the 2001 International Conference on Dependable Systems and Networks*, pages C–14 – C–16, Göteborg, Sweden, July 1–4, 2001.

[14] R. Marty. Thor - a tool to test intrusion detection systems by variations of attacks. Master's thesis, Swiss Federal Institute of Technology (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), Zurich, Switzerland, 2002. http://www.raffy.ch/projects/ids/thor.ps.gz.

[15] P. A. Porras, M. W. Fong, and A. Valdes. A mission-impact-based approach to infosec alarm correlation. In A. Wespi, G. Vigna, and L. Deri, editors, *Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *LNCS*, pages 95–114, Zurich, Oct. 16–18, 2002. Springer-Verlag.

[16] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, Oct. 7–10, 1997. National Institute of Standards and Technology/National Computer Security Center.

[17] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levitt. The design and implementation of an intrusion tolerant system. In *International Conference on Dependable Systems and Networks (DSN'02)*, pages 285 – 292, Washington, DC, USA, June 23–26, 2002. IEEE Computer Society. `http://doi.ieeecs.org/10.1109/DSN.2002.1028912`.

[18] R. F. P. (rfp@wiretrip.net). A look at whisker's anti-IDS tactics. Internet. `http://www.ussrback.com/docs/papers/IDS/whiskerids.html`. Accessed July, 2008.

[19] M. Roesch. Snort: lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pages 229–238, Seattle, Washington, Nov. 7–12, 1999.

[20] E. Tombini, H. Debar, L. Mé, and M. Ducassé. A serial combination of anomaly and misuse IDSes applied to HTTP traffic. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 428–437, Washington, DC, USA, 2004. IEEE Computer Society.

[21] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stavridou, and T. E. Uribe. An architecture for an adaptive intrusion-tolerant server. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Tenth International Workshop on Security Protocols*, volume 2845 of *Lecture Notes in Computer Science*, pages 158–178, Cambridge, UK, Apr. 17–19, 2002. Springer-Verlag. Also presented at *The International Conference on Dependable Systems and Networks (DSN 2001)*, Göteborg, Sweden, July, 2001.

[22] G. Vigna, F. Valeur, and R. A. Kemmerer. Designing and implementing a family of intrusion detection systems. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on foundations of software engineering*, pages 88–97, New York, NY, USA, 2003. ACM Press.

[23] G. White and V. Pooch. Cooperating security managers: Distributed intrusion detection systems. *Computer & Security*, 15(5):441–450, 1996.

[24] Y. Zhai, P. Ning, P. Iyer, and D. S. Reeves. Reasoning about complementary intrusion evidence. In *ACSAC'04: Proceedings of the 20th Annual Computer Security Applications Conference*, pages 39–48, Washington, DC, USA, 2004. IEEE Computer Society.

Why We Should Take a Second Look at Access Control in Unix

Jason Crampton
Information Security Group, Royal Holloway, University of London

Abstract

Unix is an operating system that began development almost 40 years ago. It has a very simple mechanism for controlling access to protected resources based on the owner-group-world model. This simple model has not attracted much interest from the access control community. In this paper we argue that the Unix access control mechanism has some interesting features of relevance to modern authorization services. We present a formal model for the Unix access control mechanism and compare its characteristics with those of role-based access control and XACML, two popular foundations for authorization services. We then discuss what lessons may be learned from the Unix model and how those lessons might be applied in the future.

1 Introduction

Access control is the process by which the interaction of users with protected resources is constrained. In the context of computer systems, it may also be referred to as *authorization*. Along with audit and authentication, authorization is one of the essential security services deployed in modern computer systems.

There has been a large amount of research in the last 30 years on authorization, much of it focusing on techniques and assumptions introduced in the protection matrix model [4, 5] and the Bell-LaPadula model [2]. Very little of that research has considered the Unix access control mechanism. In this paper, we argue that authorization in Unix has some interesting features, from which useful lessons could have been learned.

One of the problems with current research on authorization is the confusion over what is meant by authorization policy. In the next section, therefore, we attempt to make a clear distinction between authorization policy and authorization state. This distinction enables us to highlight some of the problems with role-based access control (RBAC) and XACML.

We then define a formal mathematical model for Unix access control, in the style of classic authorization models, such as those that exist for the protection matrix [5], Multics [2] and role-based access control [12]. We identify those parts of the model that are policy and those that are state. In doing this, we discover that Unix anticipates many features that have been "discovered" by researchers over the last 30 years. Based on the lessons learned from our study of Unix, in Section 4 we describe how to design an authorization service.

In Section 5, we compare RBAC and XACML to Unix and to our design programme. Roughly speaking, we conclude that RBAC is too simple and that XACML is too complex.

2 On authorization models and policies

Authorization is concerned with limiting access by users to protected resources. The limits on access are typically expressed as rules or policies. When a user makes a request to perform some

action on some resource, a policy evaluation mechanism will determine whether the request is authorized by the policy and grant access if it is (and deny it otherwise).

The protection matrix is one of the oldest concepts in access control. It is a data structure M in which rows are associated with subjects (synonymous with users), columns are associated with objects (protected resources), and the matrix entry for subject s and object o, denoted $M_{s,o}$, contains the actions for which s is authorized with respect to o.

Of course, M can be represented in a variety of ways. We could, for example, treat M as a "protection relation" $R_M \subseteq S \times O \times A$, where $(s, o, a) \in R_M$ denotes that s is authorized for action a on object o; then we have $M_{s,o} = \{a \in A : (s, o, a) \in R_M\}$. Alternatively, we could regard M as a "protection function": $f_M : S \times O \to 2^A$, where $f_M(s, o)$ corresponds to $M_{s,o}$.

In the abstract, we grant the request (s, o, a) if $a \in M_{s,o}$ and deny it otherwise. In a practical implementation, the relational approach is the most economical in terms of storage if we make the assumption that request (s, o, a) is denied whenever $(s, o, a) \notin R_M$. This is because the matrix representation includes empty entries, which are rendered redundant with respect to the above "meta-rule" for policy evaluation. Henceforth, we will tend to use R_M, rather than M.

We now make an important point about terminology. Consider the following question: Does R_M define the authorization policy? Many people would claim that it does. We would claim that it does not. Moreover, we believe that regarding R_M as a policy leads to confusion, the poor design of authorization models and weak implementation of authorization mechanisms (as we will illustrate in Section 5).

To provide some justification for our views, let us now consider the Chinese Wall policy [3]. This policy is applied in situations where objects have an owner and the set of owners can be partitioned into conflict of interest classes. The policy states that no user is allowed to view a document owned by w belonging to conflict of interest class k if the user has already viewed a document owned by $w' \neq w$ also in k. By definition, the decision whether to grant an access request can only be made by considering which documents the requester has already viewed. In its simplest form, then, the evaluation of the Chinese Wall policy is dependent entirely on system state: namely, audit information recording which documents users have previously accessed. In other words, the Chinese Wall policy defines a rule for deciding whether an access request should be granted or not.

With this interpretation of authorization policy, the protection matrix policy is simply a rule that says grant the request if and only if it belongs to R_M. We might reasonably say that R_M defines authorization state, rather than authorization policy.

To reinforce this distinction, consider an information flow policy for confidentiality (as defined in the Bell-LaPadula model [2]). The simple security property defines an authorization policy, while the security lattice L and the security function λ define authorization state.[1] We now try to formalize the distinction between policy and state.

Definition 1 *An authorization policy is a specification of a decision-making function that takes a request query and authorization state as inputs and returns an authorization decision.*

Definition 2 *An* authorization evaluation function *is an implementation of an authorization policy.*

An authorization evaluation function (AEF) for the simple security property, for example, would take two security labels $\lambda(s)$ and $\lambda(o)$ and the security lattice L as inputs and return `true` if $\lambda(s) \geqslant \lambda(o)$ (in L) and `false` otherwise.

What, then, is an access control model? A survey of the access control literature suggests that the accepted meaning of the term is an abstract mathematical description of authorization

[1] The simple security property states the read request $(s, o, \mathtt{r})$ is only granted if $\lambda(s) \geqslant \lambda(o)$ [2].

state, together with a definition of policy. The RBAC96 model provides a "textbook" example of an access control model [12]. In the next section, we develop a model for the Unix access control mechanism.

3 Access control in Unix

We start by introducing a model for access control in Unix and describing how the access control mechanism in Unix is actually implemented. This mechanism is generally regarded as being rather simplistic, but we believe that by describing a formal model for Unix we obtain some interesting insights into access control models and how models tend to be implemented.

In Unix there exists a mapping between users and security identifiers (SIDs). Every user is associated with a unique user SID (UID) and may be associated with one or more group SIDs (GIDs). Every object is associated with a unique owner UID and a unique GID. Access decisions are made on the basis of the relationship that exists between the requesting user and the target of the request.

3.1 A formal model

We assume the existence of a set of users U, a set of objects O, a set of generic actions $A = \{\mathtt{r}, \mathtt{w}, \mathtt{x}\}$, and a set of principals $P = \{\mathtt{owner}, \mathtt{group}, \mathtt{world}\}$. We define two functions:

$$\pi_1 : U \times O \to P \quad \text{and} \quad \pi_2 : P \times O \to 2^A.$$

We now define π_1. We assume there exist two sets of identifiers *SID* and *GID*, and we define functions $\alpha : U \cup O \to \mathit{SID}$, $\beta : U \to 2^{\mathit{GID}}$, $\gamma : O \to \mathit{GID}$. Then we define

$$\pi_1(u, o) = \begin{cases} \mathtt{owner} & \text{if } \alpha(u) = \alpha(o), \\ \mathtt{group} & \text{if } \gamma(o) \in \beta(u), \\ \mathtt{world} & \text{otherwise.} \end{cases}$$

In simple terms, a request (u, o, a) is granted if $a \in \pi_2(\pi_1(u, o), o)$, and denied otherwise.[2] In order to evaluate a request, then, three inputs are required: the request itself, the principal with which the requester is associated (π_1), and the authorization state (π_2).

3.2 Policy and state

Let us now try to identify what is policy and what is state in the Unix model. Checking an access request is a two-step process: first we must determine the principal for which the requester is authorized and then we must determine whether the principal is authorized for the requested action. There must, therefore, be two policies. The first policy determines the principal associated with the requester and requested object, given the SIDs of the requester and object. The state required as input to this policy, therefore, is defined by α, β and γ. The second policy determines whether a principal is authorized, given the matrix defined by π_2. In this case, the policy is just the simple protection matrix policy and π_2 represents the authorization state.

[2] In fact, checking whether a request is granted or not is slightly more complicated. All objects in Unix, from a logical perspective, are files organized within a tree-like namespace. Each object is identified by a path name within that namespace. A request by user u to access object o with pathname $/d_1/\ldots/d_m/n$ is only granted if the user is permitted to enter every parent directory d_i. Therefore, the Unix access control mechanism actually evaluates several access requests of the form $(u, d_i, \mathtt{x})$, in addition to the original request. The original request is denied if any of these (sub)requests is denied. This is merely an extension to the basic Unix authorization policy; it is required because of the implementation of objects in Unix.

3.3 Observations

We now make some observations about this model.

Remark 3 *Users are associated with what we will call "authorization-related attributes" (ARAs).*

In the Unix world, ARAs are SIDs and GIDs. In many operating systems and other computing environments where there is a centralized repository of user information, an ARA is often some form of SID. Indeed, this type of identity-based access control has been the *de facto* standard for many years, although open distributed computing is changing this.

Remark 4 *Authorization state is not necessarily expressed in terms of ARAs.*

In general, authorization state is expressed in terms of "policy-related attributes" or *principals*. In their seminal paper on the protection of information in computer systems, Saltzer and Schroeder define a principal to be "The entity in a computer system to which authorizations are granted" [11]. In Unix, these principals are `owner`, `group` and `world`. In role-based access control, for example, principals are roles. Information flow policies, however, are defined in terms of security labels, which do not correspond so well with the idea of principals. Hence, we prefer the notion of *policy related attributes* (PRAs), although we will also continue to use the term principal.

Remark 5 *There are two distinct kinds of evaluation that are required when deciding whether a request should be granted: one that decides the principal(s) with which a user is associated (π_1), and one that decides the requests for which a principal is authorized (π_2).*

π_2 represents a familiar type of access control model seen in many centralized identity-based systems. As we noted in the previous section, π_2 is nothing more than a simple protection matrix. π_1 also defines an authorization policy, by determining the principal for which a user is authorized. In modern terminology, π_1 is an attribute-based authorization policy. Specifically, π_1 associates ARAs with PRAs.

Remark 6 *π_1 is independent of π_2.*

Note that π_1 and π_2 can be interpreted as matrices. In the case of π_1 the rows are indexed by users and columns by objects and a matrix entry is a principal. In the case of π_2, the rows are indexed by principals and columns by objects, and a matrix entry is a set of access rights. This is nothing more than a simple protection matrix, in which the number of rows is fixed (since the set of principals is fixed).

Indeed, interpreting π_1 as a look-up table that associates each user-object pair with a principal, we can define any set of principals and any look-up table that we wish. Unix chooses a particularly simple set of principals and a fixed algorithmic way of specifying the entries in this table, based on the relationships that exist between a user's SIDs and an object's SIDs. But we could, for example, leave the configuration of π_1 at the discretion of the administrator. In this case, each user-object pair is associated with at most one principal by the administrator. A row in the table defines a list of object-principal pairs for a particular user (analogous to a capability list derived from a standard protection matrix). This would be used to determine which principal should be passed to π_2 for a given user and object. This type of approach could be particularly useful if wild cards are used in the definition of objects: $/d/*$, for example, would denote all objects with path name starting $/d$, and an entry $(/d/*, \texttt{owner})$ for user u would denote that u is the owner of all objects with that form of path name.

Remark 7 *π_1 need not be defined in terms of SIDs.*

We note that different types of user attributes and/or environmental attributes could have been used to specify $\pi_1(u, o)$. We could, for example, include time of access as a parameter to π_1, so that access is determined by the time of the request as well as the relationship that exists between user and requested object. Similarly, location could be a parameter to π_1.

Remark 8 *The user is not bound to a principal until an access request is evaluated.*

The late binding of requester to principal(s) and the fact that π_1 need not be defined in terms of SIDs means that the Unix access control model could be used to define so-called *context-sensitive* or *context-aware* authorization policies (see [7], for example). In particular, context-sensitive principals (for example, `internal`, `external`, etc), where access is (in part) determined by environmental conditions (such as user location or time of request), can be defined using a policy analogous to π_1.

Arguably, the late binding of requesters to principals and the independence of π_1 and π_2 makes access control in Unix easier to manage, more modular and more flexible. In particular, the number of principals used in the authorization policy can be reduced.[3]

Remark 9 *A typical implementation of the Unix access control model is quite different from the model.*

Note that we can interpret $B \subseteq A$ as an element of $\{0,1\}^3$, where $b_r b_w b_x \in \{0,1\}^3$ is the characteristic set of B. The set $\{r, x\}$, for example, is represented by 101. This, in turn, means that any column of the matrix representing π_2 can be represented as an element of $\{0,1\}^9$. This *permission mask* can be associated with the object corresponding to that column. Each group of three bits in the permission mask $\mathtt{r}_o\mathtt{w}_o\mathtt{x}_o\mathtt{r}_g\mathtt{w}_g\mathtt{x}_g\mathtt{r}_w\mathtt{w}_w\mathtt{x}_w$ corresponds to the permissions associated with each principal for that object.

Recall that deciding whether a request should be granted requires mapping the user to a principal using π_1 and then deciding whether the principal is authorized given the state encoded in π_2. In general, it is not practical to store and manage the authorization state encoded by π_1 and π_2 centrally. In Unix, π_1 is computed using α, β and γ which associate SIDs and GIDs with users and objects. The functions α and β and γ are implemented partly in the `passwd` file (for users) and partly in the metadata of each object. The state π_2 is implemented using fixed-length access control lists (ACLs), each of which is represented as a 9-bit permission mask associated with the appropriate object. To decide a request, the relevant authorization state (the ACL for the requested object) is passed to an evaluation function, along with the relevant principal (determined by π_1).

4 Building an authorization service

Let us now draw some conclusions from our examination of access control in Unix. In particular, we will try to identify what problems need to be solved in order to construct an authorization service.

The first problem to be addressed is the model for associating PRAs with authorized requests. In particular, we must define what type of PRAs will be used. This might be SIDs, roles, security labels, or a fixed set of principals (as in Unix). The association of PRAs with

[3] Of course, some people might also argue that Unix is limited in the authorization policies that it can express. Indeed, newer versions of Unix include support for more complex ACLs (`http://www.freebsd.org/doc/en/books/handbook/fs-acl.html`)

authorized requests may be "discretionary", as in Unix, where $\pi_2(p, o)$ is determined by the owner of each object (or `root`). Or it may be "mandatory": in Multics, the authorizations of a security label are determined by the simple security property and the *-property.

The second issue is the model for associating ARAs with PRAs. This may be "mandatory", as in Unix, where $\pi_1(u, o)$ is determined by the respective identifiers associated with u and o.[4] Or it may be "discretionary": in Multics, for example, ARAs (subjects) are associated with a PRA (a security label) by the administrator.

The third problem concerns the evaluation of policies for the models defined in the previous two steps. This is really a question of designing two authorization evaluation functions, one to evaluate π_1 (to identify the PRAs that should be associated with the requester) and one to evaluate π_2 (to decide whether the request is authorized), which we denote aef_1 and aef_2, respectively. In many cases, the function aef_1 is very simple (as in Unix, for example). However, in open distributed systems, such a function may be non-trivial (see RBTM [6], for example). We will need to refer to the current authorization state to evaluate either function.

The definitions of aef_1 and aef_2 will include the inputs that each function requires. Hence, we need to decide how to implement the authorization state defined in the two models, and how that state (or parts thereof) will be presented to the decision function in the form of input parameters. In the case of Unix, the authorization state input to aef_1 is implemented by α, β and γ, and the authorization state is defined by π_2 and is implemented in a decentralized way as a set of object permission masks.

Finally, we need to decide how the inputs for aef_1 and aef_2 will be acquired. This is very simple in the case of systems in which there exists a centralized repository of information about users and objects. In systems where there is no centralized database for users – so-called "open" systems – the problem of acquiring authentic ARAs is considerably more complex, requiring some form of "trust infrastructure" usually implemented using cryptographic techniques. In Unix, for example, the authentication service supplies information about α, β and γ for subjects, while the file service provides similar information for objects. The authorization evaluation function can trust each of these components to supply authentic information about ARAs (and hence users) and objects.

Hence, the last (and in some sense the first) question that needs to be addressed when designing an authorization is how an authentic binding between human users and ARAs can be achieved. Typically authentication in computer systems serves to confirm that a human being is who she claims to be and to bind that human identity to data or attributes that can be processed by the computer system. The authentication service in modern operating systems, for example, creates a shell or desktop process for an authenticated user and associates one or more (security) identifiers with that process. Unix, in particular, associates an SID and one or more GIDs with an authenticated user. This information is typically collocated with the username and password for the user in the `passwd` file. However, in open systems, it is necessary to acquire user-ARA bindings from third parties. SAML assertions are perhaps the most obvious implementation of user-ARA bindings [8].

To summarize, there are five fundamental considerations when designing an authorization service. Recall that for the purposes of this paper, a "model" includes a logical description of authorization state and a means of evaluating requests.

(1) Define the PRA-request authorization model by which principals are associated with authorized requests.

(2) Define the ARA-PRA authorization model by which ARAs are associated with principals.

[4] Of course, the binding of users to ARAs in Unix is "discretionary", in that it is decided by the system administrator (`root`).

(3) Define the functions that will implement the PRA-request model and ARA-PRA model.

(4) Define the actual authorization state that will be used to implement the logical authorization state.

(5) Define the mechanisms by which authentic user-ARA bindings will be acquired.

We note that in closed systems, it will be more usual to decide how authorization state will be implemented before designing the AEFs. That is, steps 3 and 4 will be reversed. However, in open systems, we cannot assume anything about the applications that may wish to use the AEFs we define. (The XACML policy decision point, for example, is intended for use by multiple policy enforcement points (PEPs), each "guarding" different resources.) In this situation, we must ensure that the AEFs can process authorization requests from a wide variety of PEPs.

The priority, therefore, is to design the interfaces that the AEFs expose to those PEPs, so that the PEPs can form appropriate authorization requests. It is here that we advocate a completely different approach from XACML. In XACML, it is assumed that the policy decision point (PDP) fetches or "pulls" the appropriate policy from some policy repository (the policy administration point in XACML jargon). But it is not at all clear how the PDP identifies the appropriate policy. Indeed, there seems to be some confusion within the XACML community as to what "appropriate" means or even if it is possible to identify appropriate policies.[5] We believe that the relevant authorization state should be "pushed" as input parameters to the appropriate AEF. By defining the input types expected by the AEF, the PEP is able to construct an appropriate authorization request.

5 A critique of current research

In this section we examine RBAC96 [1], RBTM [6] and XACML [9]. RBAC96 is a well-known and widely cited role-based access control model. RBTM stands for role-based trust management: it is intended for use in open distributed environments, and is also widely cited. XACML is a standard that defines a syntax for authorization policies and a specification of a policy decision point for evaluating access requests. XACML has attracted considerable interest from industry and the research community in recent years. We argue that RBAC96 and XACML do not represent much of an advance on the basic Unix model; indeed, in some respects they are actually rather more primitive. RBTM, in contrast, does have some interesting and useful features, but only provides half a solution.

5.1 Role-based access control

ANSI-RBAC [1] is a standardized version of RBAC96 [12], in which the central concept is a *role*. The basic idea is that users and permissions (object-action pairs) are associated with roles, and that there are significantly fewer roles than there are users or permissions. The motivation for RBAC is to reduce the effort required to (i) manage access control policies and (ii) to answer questions of the form "What is user x authorized to do?". The latter question is difficult to answer in access control mechanisms based on ACLs (since it requires an examination of each ACL to see whether x is authorized for any form of interaction with the associated object).

5.1.1 Flat RBAC

More formally, *flat RBAC* defines two relations $UA \subseteq U \times R$ and $PA \subseteq P \times R$, where U, R and P are the sets of users, roles and permissions, respectively. The user-role assignment relation

[5] `http://lists.oasis-open.org/archives/xacml-dev/200804/msg00010.html`

(UA) associates users with roles and the permission-role assignment relation (PA) associates permissions with roles

A user u is authorized for a permission $p = (o, a)$ if there exists a role r such that $(p, r) \in PA$ and $(u, r) \in UA$. Note that we can rewrite PA as $PA' \subseteq R \times O \times A$, where $(r, o, a) \in PA'$ if and only if $((o, a), r) \in PA$. Similarly, PA can be written as a matrix with rows and columns indexed by roles and objects, respectively, where action a belongs to the matrix entry for r and o if and only if $((o, a), r) \in PA$.

Let us consider this from the perspective of our Unix model, where π_1 defines a mapping from users and objects to principals and the function π_2 defines a mapping from principals and objects to authorized actions. Seen from this point of view, both ARAs and principals in RBAC are roles, UA defines π_1, while PA defines π_2. Flat RBAC – with its strong coupling of ARAs and principals, and the simple lookup tables UA and PA – seems like a rather simplistic model, when compared to Unix.

Let us now examine RBAC as a model from which we might try to build an authorization service. Obviously, the PRA-request authorization model is represented by PA and the ARA-PRA authorization model is represented by the identity function (since roles are both ARAs and PRAs). The user-ARA bindings are defined in UA, which will (presumably) be implemented in a way that is analogous to the association of users with SIDs and GIDs in Unix. The function that implements the PRA-request model is simply one or more look-ups in PA. So the remaining question is how logical authorization state will be realized; in other words, how will PA be implemented in practice.

As we noted above, PA is just an alternative encoding of a protection matrix, the only advantage of this representation being that empty cells of the matrix need not be recorded. It is well known that a protection matrix is to generally too large to implement in practice, and the same presumably applies to the alternative relational representation. Given that most operating systems use ACLs and associate them with objects, rather than use a single relation encoding the protection matrix, it seems likely that the PA relation may well be encoded as ACLs in a real implementation. Of course, this means that it now becomes difficult to answer the question "What is role x authorized to do?" (for precisely the same reason that we cited earlier).

One of the claims of the supporters of RBAC is that it makes management of authorization state easier. This claim is certainly not supported by a comparison with Unix. Suppose that we wish to apply either the RBAC model or the Unix-based access control (UBAC) model to a particular scenario. Let us assume that the number of roles is equal to the number of Unix groups. Then the "cost" of associating users with ARAs is the same. The cost of managing the association of principals with requests, however, is much lower in UBAC, because there are only three principals. Moreover, the cost of evaluating an access request is likely to be lower in UBAC, because it is only necessary to compute the correct principal for the user-object pair and then find the appropriate part of the permission mask for the object. In contrast, even in flat RBAC, it will be necessary to check a number of entries in PA (equivalently, we may need to check a number of different entries in an ACL). In short, it is difficult to see what advantages RBAC offers over ACL-based systems that provide support for groups.

5.1.2 Other models

Hierarchical RBAC includes the role hierarchy relation $RH \subseteq R \times R$, where (R, RH) is an acyclic directed graph. The transitive reflexive closure of RH defines a partially ordered set. We write $r \leqslant r'$ to denote that there exists a path from r' to r. In hierarchical RBAC, u is authorized for p if there exist roles r and r' such that $(p, r) \in PA$, $(u, r') \in UA$ and $r \leqslant r'$. In

this case, π_1 is defined by *UA* and *RH*, while π_2 is defined by *PA* as before. The only difference here is that additional information needs to be recorded for *RH*.

Role-based trust management (RBTM) provides a powerful model for associating users with roles in open distributed environments [6]. However, RBTM is only designed to map users to principals; it is silent on the subject of mapping principals to authorized requests (presumably because deciding access requests in a role-based model is simple once the set of roles is known).

5.2 XACML

XACML policies are collections of rules. Each rule defines a set of subject-object-action triples, and an effect (allow or deny). Rules are gathered together in policies and policies may be grouped in policy sets. The effect of a policy or a policy set is determined by the rule-combining and policy-combining directives (such as deny-overrides) defined in the policy and policy set, respectively.

A rule can be encoded as a four-tuple $(\mathbf{s}, \mathbf{o}, \mathbf{a}, e)$, where $\mathbf{s}$ defines some subset of users, $\mathbf{o}$ defines some set of objects, $\mathbf{a}$ defines some subset of actions, and e is the effect. The tuple $(\mathbf{s}, \mathbf{o}, \mathbf{a})$ is called the *target*. In crude terms, an XACML policy set is nothing more than an encoding of a protection matrix, where $\mathbf{s}$ defines the row label and $\mathbf{o}$ defines the column label. In general, $\mathbf{s}$ can be thought of as a set of ARAs. In the RBAC profile of XACML, for example, $\mathbf{s}$ is a role attribute of the requesting user. We believe the interesting part of XACML, therefore, is how to determine whether a rule with target $(\mathbf{s}, \mathbf{o}, \mathbf{a})$ is relevant to a request (u, o, a). (Roughly speaking, this corresponds to the problem of defining π_1.) In general, it is necessary to seek additional information about u in order to determine the ARAs associated with u. The collection of this information is outside the scope of XACML.

Let us now compare XACML with our programme for constructing an authorization service. The PRA-request model is defined by the schema for XACML policies. However, arbitrary principals are permitted. We believe that it is important to define the principals that are used to define authorization state. It is not possible to define the ARA-PRA model otherwise. As we noted above, the ARA-PRA model is not defined at all for XACML, although there is a SAML profile for XACML [10], which facilitates the exchange of authorization assertions.

In contrast, the function for deciding requests is the XACML policy decision point (XPDP). In our terminology, there is only one policy, the one implemented by the XPDP, and it is applied to *all XACML policies*. We believe this to be the fundamental mistake made by the designers of XACML, and one that arises from a blurring of the distinction between policy and state in XACML. An XACML "policy" is mainly authorization state, essentially encoding a protection matrix. However, the "policy" also contains directives about how to process the effects of rules. This is more like what we mean by policy. In short, the XPDP attempts to act as a "universal authorization evaluation function", taking both state and policy as inputs (although the XACML standard certainly does not phrase it in these terms). There is no good reason to suppose that it is possible to define such a universal function; indeed, such a function would resemble a universal Turing machine.

An obvious question to ask is how XACML would represent Unix authorization state and access control policy? The natural way of doing it would be to define a policy for each object, each policy containing three rules, one for each of the Unix principals and the actions for which each of them is authorized. This, of course, illustrates the point that XACML does not provide an answer to the question "How do we map the requester to a subject attribute (a Unix principal in this case)?". Of course, in a closed system, XACML can acquire these attributes from the authentication service in the same way as in Unix. But the point we are making is that XACML by itself cannot be used to build an authorization infrastructure. An alternative rendering of

the Unix mechanism in XACML would be to ignore principals altogether, so that each rule has the form user u can perform action a on object o.

XACML has a single evaluation function (the XPDP), which means that all authorization policies have to be encoded in authorization state. Therefore, it is doubtful whether XACML is capable of handling "stateful" authorization policies. Consider the following access control requirement: *in any state, if a subject has simultaneous "observe" access to* o_1 *and "alter" access to* o_2*, then* $\lambda(o_1) < \lambda(o_2)$. This is the well known *-property – designed to prevent illegal indirect information flows – quoted in its original form from the seminal paper by Bell and LaPadula [2, page 17]. This requirement implies that before any read (respectively, write) access request can be granted it is necessary to consider all objects for which the requesting subject currently has write (read) access. In other words, deciding whether the *-property would be violated by a request requires information about the run-time state of the computer system, as well as the respective security labels of the subject and object.[6] It is very difficult to see how XACML could enforce such a requirement.

Quite apart from the fact that XACML is not able to encode a number of important authorization policies, it seems unlikely that XACML is a practical foundation for an authorization infrastructure: in particular, XACML policies will be difficult to author and maintain. Moreover, proving that an XACML policy actually encodes the desired enterprise security requirements is rather difficult, particularly in comparison with existing models such as the protection matrix, RBAC and Bell-LaPadula. Finally, modifying XACML policies to reflect changes in enterprise security policy may well be a significant undertaking. Suppose, to take a contrived example, that we wished to move from an information flow policy for confidentiality written in XACML to an information flow policy for integrity. Then we would need to completely re-write the XACML policy in order to encode the new authorization logic as well as authorization state, because XACML policies are always processed by the same AEF (the XPDP).

6 Conclusion

In this paper we have examined some basic questions concerning authorization, particularly the distinction between authorization policy and authorization state. We have developed a formal model for Unix and illustrated the importance of these questions. We then described the problems that need to be addressed when building an authorization service. Finally, we discussed RBAC and XACML in the context of the ideas presented in the earlier sections. The main contributions of the paper are:

- to make clear the distinction between authorization models, authorization policies, authorization state and authorization evaluation functions, and to demonstrate why such distinctions are important;
- to develop a model for Unix access control and to describe its relevance to the construction of any authorization system;
- to identify two distinct types of authorization policy, one mapping users to PRAs and one mapping PRAs to authorized actions;
- to observe that approaches such as RBAC and XACML are unlikely to be sufficiently versatile to be used as the basis for authorization services; and
- to outline a new approach to policy evaluation.

[6] In fact, the satisfaction of the *-property in the Multics implementation of the BLP model is implied by enforcing stronger conditions that can be checked without reference to the run-time state [2, pages 49–53].

While we do believe that the motivation for developing XACML is absolutely correct – there does need to be a way for distributed heterogeneous computer systems to exchange authorization-related information – we do not believe that the XACML standard is the correct solution. (As can be seen from our remarks in Section 5, we are even less convinced that RBAC provides an appropriate solution.) XACML tries to provide a single platform- and policy-independent authorization evaluation function (the XACML PDP) as well as a way of encoding authorization state; we have doubts about whether this is even possible, but we also believe it addresses the wrong issue. Given the observations in this paper, we believe that a more appropriate question to try to answer would be how to pass authorization state between authorization evaluation functions (AEFs). In our approach, each AEF is a function with a signature that defines the input types expected by the AEF (in addition to the request itself). In practice, this signature would be public and could, for example, be represented using XML, or even XACML, syntax. The AEF would process requests with inputs that conform to its signature. These inputs could be supplied in XML format, perhaps using the XACML request context syntax, for example. More complex AEFs could be constructed by co-ordinating the outputs from two or more simpler AEFs. Hence, we could implement separate AEFs for the simple security property, the *-property and the protection matrix, and combine their outputs to provide an AEF that evaluates requests with reference to policy defined in the Bell-LaPadula model.

In other words, there is plenty of scope for future work. Our first priority is to develop a new architecture for authorization based on the ideas in this paper. This architecture will identify the components required by an authorization service in open distributed systems and the protocols that need to be supported by each of those components. The next phase of the work will be to develop a syntax and semantics for messages to implement those protocols.

Acknowledgements The author would like to thank the anonymous reviewers for their constructive remarks, which have helped to improve the content and presentation of the paper.

References

[1] American National Standards Institute. *ANSI INCITS 359-2004 for Role Based Access Control*, 2004.

[2] D.E. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, Mitre Corporation, Bedford, Massachusetts, 1976.

[3] D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[4] G.S. Graham and P.J. Denning. Protection – principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 417–429, 1972.

[5] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[6] N. Li, J.C. Mitchell, and W.H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[7] P.D. McDaniel. On context in authorization policy. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies*, pages 80–89, 2003.

[8] OASIS. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, 2005. OASIS Standard (S. Cantor, J. Kemp, R. Philpott, and E. Maler, editors).

[9] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Committee Specification (T. Moses, editor).

[10] OASIS. *SAML 2.0 Profile of XACML v2.0*, 2005. OASIS Standard (A. Anderson and H. Lockhart, editors).

[11] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 36(9):1278–1308, 1975.

[12] R. Sandhu, E.J. Coyne, H. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

From policies to aspects in KLAIM

Luke Herbert and Einar Egilsson
Department of Informatics, Technical University of Denmark
{lth,ee}@imm.dtu.dk

September 23, 2008

Abstract

The aspect oriented programming paradigm facilitates the separation of cross cutting concerns in system development. Security policies are a typical such concern and in this paper we present a simple policy language, LUNAR, and show how it can be translated into aspect definitions. We perform the development for KLAIM, a small kernel language for agent interaction and mobility, and show how static analysis can be used to reduce the number of aspect definitions. This technique has been applied to a larger case study, namely the electronic invoice system at DTU.

1 Introduction

Motivation Modern business processes can be extremely complicated and the need exists to verify that these processes comply with a given security policy [H05]. This situation might arise in an invoicing system, where the processes put in place could allow embezzlement or other undesirable activity due to loopholes in the security policy, or in the business process itself. Today, the flow of many business processes is dictated by the software they employ, software which has often been required to quickly evolve as the needs of the business change, and which has not been built with security in mind.

The separation of concerns inherent in aspect oriented programming [MK03] allows security features to be developed separately from the main body of code, thus removing the complexity of adding such features and allowing them to change as needed. In addition, security features can be developed separately by people specialized in this field, hopefully ensuring a higher quality of code. However, the separation of code into aspects and the transformation of a security policy into aspects present many new challenges.

Contribution In Section 2 we introduce the subset of KLAIM to be used for specifying business processes. KLAIM is a kernel language for distributed computation where data and processes can be moved from one computing location to another [NFP98]. Processes can interact with one another by performing input, output and read operations and in Section 3 we present a simple policy language called LUNAR that will be used to limit which operations can be performed at various stages in a workflow process. To enforce the policies we present in section 3.1, we make use of an aspect oriented version of KLAIM, AspectK, developed in [HNNY08]; in Section 4 we present a slight extension of the AspectK language together with an algorithm for transforming LUNAR policies into aspects; a subsequent weaving will then embed the aspects into the code itself. Finally, in Section 5 we present a static analysis that, given a KLAIM workflow process and a set of LUNAR policies will identify which policies might be violated, as it is only necessary to construct aspects for those. In Section 6 we discuss related work and in Section 7 we give concluding remarks.

A overview of the basic design of LUNAR is shown in Figure 1.

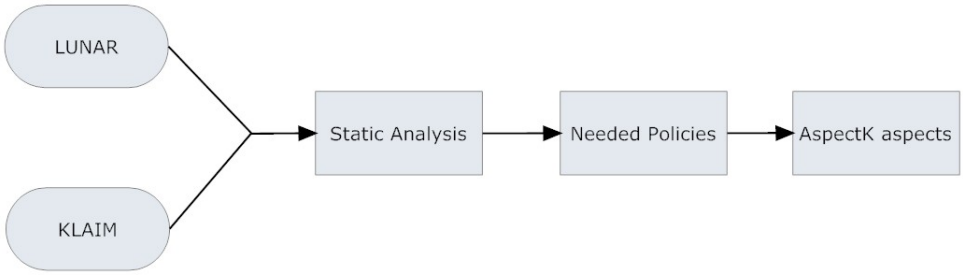


Figure 1: The design of the LUNAR system

1.1 Case Study: the DTU invoice system

The above development has been carried out for a larger example modelling the workflow of the electronic invoice system at DTU; specifically how this system is used within the department of Informatics and Mathematical Modelling. Figure 2 shows the basic flow of the invoice process.

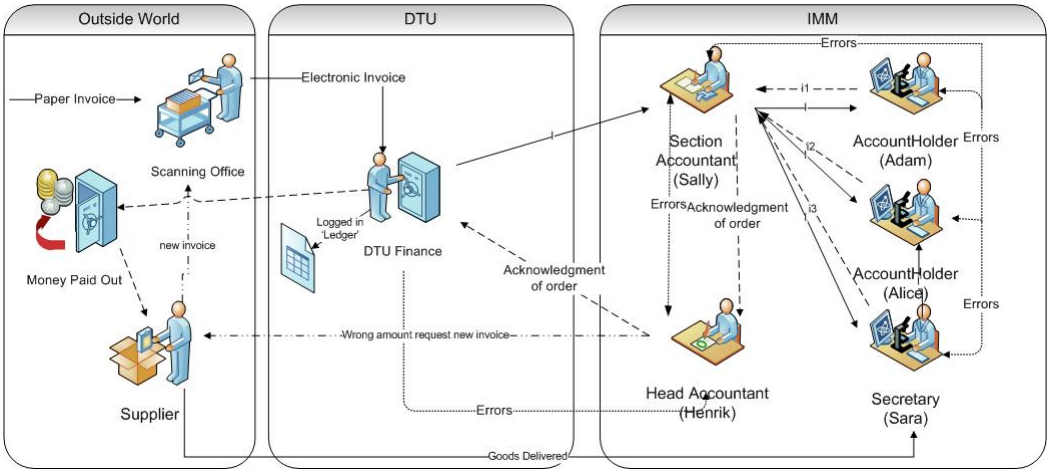


Figure 2: Overview of how the DTU invoice system interacts with the IMM department

The typical flow of an invoice through the system is described below, this is the straightforward case of a purchase being approved with no problems arising:

1. A supplier sends a paper invoice to the scanning office which scans it to create an electronic invoice.
2. The scanning office forwards the electronic invoice to DTU's main finance department.
3. The finance department forwards the invoice to the section accountant of the relevant department.
4. In this business process the section accountant generally knows which person inside the department has been ordering which things, and so they forward the invoice to the person who placed the order.
5. The person who ordered the product(s) confirms the invoice by adding their signature to it and then forwards the invoice to the relevant account holder.
6. The account holder approves the purchase by adding the correct account number to the invoice and sends it back to the section accountant.
7. The section accountant confirms that the invoice is correctly filled out, and then forwards it to the head accountant.
8. The head accountant confirms the invoice and forwards it to DTU Finance.

$N \in \mathbf{Net}$	N	::=	$N_1 \,\|\|\, N_2 \mid l :: P \mid l :: \langle \overrightarrow{l} \rangle$	
$P \in \mathbf{Proc}$	P	::=	$P_1 \mid P_2 \mid \sum_i a_i.P_i \mid *P \mid$ if $cond$ then P_1 else P_2 fi	
$cond \in \mathbf{BExp}$	$cond$	::=	$\ell_1 = \ell_2 \mid test(\overrightarrow{\ell^\lambda})@\ell$	
$a \in \mathbf{Act}$	a	::=	$\mathbf{out}(\overrightarrow{\ell})@\ell \mid \mathbf{in}(\overrightarrow{\ell^\lambda})@\ell \mid \mathbf{read}(\overrightarrow{\ell^\lambda})@\ell$	
$\ell, \ell^\lambda \in \mathbf{Loc}$	ℓ	::=	$u \mid l$	$\ell^\lambda ::= \ell \mid !u$

Table 1: KLAIM Nets and Processes Syntax

9. Finance pays the invoice and enters the amount into the main DTU ledger.

When errors or malicious activity occurs, other flows through the system take place. Some examples of alternative flows are described below:

- The section accountant sends an invoice to the wrong person. That person does not confirm the purchase but sends it back to the section accountant, who will then have to find out who the correct recipient is and forward it to them. In practice such errors accumulate to large amounts of wasted time and could reveal a secret purchase, such as a birthday present.
- If the buyer sees that the price on the invoice is wrong, he can send a complaint to the supplier and wait for a credit note to be issued by the supplier, before sending both the invoice and credit note back to the section accountant. The invoice and credit note then both follow the normal path to get paid.
- If no one within the department recognizes an invoice payment, then it is refused and sent back to the supplier.

In this system a security policy can be enforced by controlling the flow of the invoice through the system. This is achieved by having each invoice carry a status tag which changes as it passes through the system. Each entity in the process should only receive invoices with the appropriate status. For example the scanning office should only receive *sent* invoices, DTU's finance department should only receive *presented* or *accounted* invoices and account holders should only receive *presented* or *confirmed* invoices. Other entities have similar restrictions which will be explained in detail in Section 3.1.

2 The KLAIM subset used in LUNAR

The LUNAR system analyses a subset of the KLAIM language [NFP98] similar to what is used in [HNNY08]. The syntax is defined in Table 1. A net N is a parallel composition of located processes or located tuples. Each person or entity in the business process is represented as a location. For a given entity, the collection of tuples present at its location is known as its tuple space, and the processes located at its location define its behaviour. A given location may have many processes running in parallel; each process can run just once or repeatedly, as indicated by the $*$ prefix. A process is made up of one or more actions and we allow just three operations, **in**, **out** and **read**. The **in** operation removes a tuple from a tuple space, **out** outputs a tuple to a tuple space and **read** reads a tuple from a tuple space but does not remove it. An operation parameter can be a location constant l, variable binding $!u$ or a variable u. A variable is defined when it is bound. All communication between entities in the business process is modelled by locations sending and receiving tuples.

The **read** and **in** operations attempt to match a tuple to the tuple space they are reading from. The number of parameters in the operation must match the number of elements in a tuple for it to be selected.

```
DTUFinance :: <INVOICE, PRESENTED, 500, Fona, IMM, Printer, 200>
|| DTUFinance ::

   * in(INVOICE, PRESENTED, !nr, !supplier, !department, !item, !price)@self
   . read(!accountant, department, SectionAccountant)@Staff
   . out(INVOICE, PRESENTED, nr, supplier, department, item, price)@accountant
   |
   * in(INVOICE, ACCOUNTED, !invnr, !supplier, !department, !item, !price, !confirmedBy,
        !accountNr, !project)@self
   . out(INVOICE, PAID, invnr, supplier, department, item, price, confirmedBy, accountNr,
         project)@Ledger
   |
   * in(INVOICE, REFUSED, !invnr, !supplier, !department, !item, !price)@self
   . out(INVOICE, REFUSED, invnr, supplier, department, item, price)@supplier

|| Staff :: <Sally, IMM, SectionAccountant>
|| Staff :: <Adam, IMM, AccountHolder>
|| Staff :: <Alice, IMM, AccountSupervisor>
|| Staff :: <Sara, IMM, Secretary>
|| Staff :: <Henrik, IMM, HeadAccountant>
```

Figure 3: **Code Example 1:** DTUFinance location KLAIM code

Furthermore, for any variable or constant parameter p_n in the operation, the element e_n in a tuple must have the same value for the tuple to be selected. This is used to conditionally select tuples from a tuple space.

In order to model conditional behaviour we have added a conditional expression to the language; in particular we needed to model different behaviours depending on the existence of a tuple at a particular location. The if statement can check for the existence of a tuple using the test function. The statement if $test(\overrightarrow{\ell^{\lambda}})@\ell$ then P_1 else P_2 fi will execute the P_1 process if there exists a tuple at ℓ that matches $\overrightarrow{\ell^{\lambda}}$, if the tuple does not exist process P_2 is executed instead.

A more precise definition of the language and its semantics, including its structural congruence and reaction semantics on closed nets can be found in [HNNY08].

2.1 The DTU invoice system expressed in KLAIM

For reasons of space we can not show the entire DTU invoice model here but include a representative example. The example shows the DTUFinance and Staff locations. Staff is a simple database mapping people to departments and roles, it has no processes. DTUFinance is the main DTU finance department which receives invoices and forwards them to other locations within the system.

Looking at DTUFinance in the code snippet above we see that it has one tuple located in its tuple space. The tuple format is $\langle$*type, status, invoice nr, vendor, department, item, price*$\rangle$. The DTUFinance location has three parallel processes. The first process removes an invoice tuple with the status `PRESENTED` from DTUFinance's tuple space, looks up the relevant department's section accountant by doing a **read** operation at the Staff location, and then outputs the tuple into the section accountant's tuple space. The second process pays invoices that have the status `ACCOUNTED` by removing them from DTUFinance's tuple space and outputting them to Ledger with status `PAID`. The third process removes `REFUSED` invoice tuples and sends them back to the supplier that they originated from.

In the code example above the first process finds a match since it looks for invoices with the status `PRESENTED` and then forwards the invoice to the relevant location. Neither the second nor third processes find any tuples matching their **in** operations in DTUFinance's tuple space. Other entities within our system are modelled in a similar way.

$SP \in$ **SecPolicy**	SP	::=	$A\ G$
$A \in$ **Abbreviation**	A	::=	$\$a = S \mid A\ ;\ A \mid \varepsilon$
$G \in$ **Group**	G	::=	$"name"\ P \mid G\ ;\ G$
$P \in$ **Pattern**	P	::=	$P\ ;\ P \mid V{::}\mathbf{out}(\overrightarrow{V})@V \mid V{::}\mathbf{in}(\overrightarrow{V})@V \mid V{::}\mathbf{read}(\overrightarrow{V})@V$
$V \in$ **Value**	V	::=	$S \mid [S]$
$S \in$ **Set**	S	::=	$S + S \mid t$
$t \in$ **Terminal**	t	::=	$l \mid \$\mathrm{a} \mid\ *\ \mid\ ...$

Table 2: LUNAR Syntax

3 The LUNAR policy language

LUNAR allows the user to express security policies using the LUNAR policy language, which is simple formal policy language inspired by regular expressions [LP97]. The syntax for the language is shown in Table 2. A given security policy is made up of zero or more *abbreviations* and one or more *pattern groups*.

Abbreviations allow a large set of values to be represented concisely and can be used in more than one place in a security policy. They are defined before any patterns and cannot be redefined later in the policy. Abbreviations start with a $ character to distinguish them from ordinary values. An example of an abbreviation is:

```
$AccountHolders=Adam+Alice
```

Pattern groups are defined after the abbreviations. Each pattern group is comprised of a name and one or more patterns. The patterns themselves are similar to KLAIM statements, each pattern is made up of a sender, an operation, one or more parameters, and a receiver.

$sender :: op(p_1, p_2, ...p_n)@receiver$

Each component of the pattern (*sender*, *op*, p_n, *receiver*) can be expressed as a single value or as a set of possible values for that component. A set is a list of the possible values separated by +, so for instance the *sender* part of the pattern could have the value Adam+Henrik+Sara. Pattern components are matched against action parameters to see if the pattern applies to that action, except in the case where a component is surrounded by [ ], in that case the component represents allowed values for the actions' parameter. The * token is a wildcard character matching any possible value, and the ... token can be thought of as one or more *'s, and may only be the last parameter in a parameter list. This is introduced to handle the case where tuples gain additional elements in the course of the workflow, yet we still require that the pattern matches regardless of the number of elements.

When multiple patterns apply to an action we compose the policies using the following rule: for the set of patterns P that match an action we require that all p_i in P are satisfied, therefore the listing of the patterns in LUNAR is permutation invariant.

3.1 Case study: A security policy for the DTU invoice system

The purpose of our security policy is to prevent information leaks, that is to say invoices reaching unintended recipients. The policy we present could also be classified as a business rule, however we use the term security policy as more general policies could be applied using the LUNAR system. The policy

can be enforced by controlling the flow of invoices through the system using the status tag. Specifically we create LUNAR patterns which specify the states that invoices and credit notes may be in when they are output into a particular locations tuple space. The LUNAR policy language allows for role based access control [G99], to implement this we start by creating abbreviations that define the roles within the system.

```
$Vendors=HP+OfficeSupplies+TaxiStation
$SectionAccountant=Sally
$Secretary=Sara
$AccountHolders=Adam+Alice
$HeadAccountant=Henrik
```

Using these roles, expressed as abbreviations in LUNAR policy notation, we can define the following security policy:

1. **Vendor allowed content:** Vendors may only receive complaints, invoices and creditnotes. Specifically the invoices may only have the status *sent* or *refused* and creditnotes may only have the status *sent*.
2. **ScanOffice allowed content:** The scanning office may only receive invoices and creditnotes and they may only have the status *sent*.
3. **DTUFinance allowed content:** DTU's finance department can only receive invoices and creditnotes. Specifically the invoices may only have the status *presented*, *accounted* or *refused* and creditnotes may only have the status *presented* or *accounted*.
4. **Ledger allowed content:** Only DTU's finance department may output tuples into the ledger. The tuples may be either invoices or creditnotes and they may only have the status *paid*.
5. **SectionAccountant allowed content:** Section accountants may only receive invoices and creditnotes and they may only have the status *accounted* or *presented*.
6. **Secretary allowed content:** The secretary may only receive complaints and invoices. Specifically the invoices may only have the status *presented* or *wrong* and creditnotes may only have the status *presented*.
7. **AccountHolder allowed content:** Account holders may only receive complaints and invoices. Specifically the invoices may only have the status *presented*, *confirmed* or *wrong* and creditnotes may only have the status *presented* or *confirmed*.
8. **Head Accountant allowed content:** The head accountant may only receive complaints and invoices with the status *accounted*.

The above rules are each mapped into a pattern group consisting of one or more patterns. Generally the pattern groups start with a line defining what types of tuples are allowed and subsequent lines restrict which states each type of tuple may be in when it is output into the entity's tuplespace.

Next we present two examples of rules expressed as LUNAR pattern groups. Rule 4 only allows DTUFinance to output paid invoices and creditnotes into the Ledger. This stops employees from trying to slip paid invoices past the finance department. This can be expressed as a single pattern since the invoice and creditnote have the same allowed states.

```
"4.  Ledger allowed content"
[DTUFinance]::out([INVOICE+CREDITNOTE], [PAID], ...)@Ledger
```

A more complex pattern group is required to implement rule 7. In this case the invoice and creditnote have different allowed states, the first pattern restricts the type of tuples that can be sent to account holders, and the second and third pattern restrict the allowed states for invoices and creditnotes respectively.

$S \in \mathbf{System}$	S	::=	$\mathbf{let}\ \overrightarrow{asp}\ \mathbf{in}\ N$
$asp \in \mathbf{Asp}$	asp	::=	$A[cut] \triangleq body$
$body \in \mathbf{Advice}$	$body$	::=	$\mathbf{case}\ (cond)\ sbody\ ;\ body \mid sbody$
	$sbody$	::=	$as\ \mathbf{break} \mid as\ \mathbf{proceed}\ as$
$as \in \mathbf{Act}^*$	as	::=	$a.as \mid \varepsilon$
$cond \in \mathbf{BExp}$	$cond$	::=	$\mathbf{test}(\overrightarrow{\ell^\lambda})@\ell \mid \ell_1 = \ell_2 \mid cond_1 \wedge cond_2 \mid \neg\ cond$
$cut \in \mathbf{Cut}$	cut	::=	$\ell :: a$
$\ell^\lambda \in \mathbf{Loc}$	ℓ^λ	::=	$\ell \mid !u \mid ?u \mid \ ...$

Table 3: AspectK syntax

```
"7.  AccountHolder allowed content"
*::out([INVOICE+CREDITNOTE], ...)@$AccountHolders
*::out(INVOICE, [PRESENTED+CONFIRMED+WRONG], ...)@$AccountHolders
*::out(CREDITNOTE, [PRESENTED+CONFIRMED], ...)@$AccountHolders
```

4 Conversion of policies to aspects

An aspect oriented coordination language, AspectK, was introduced in [HNNY08]. This language is a superset of the KLAIM like language we have used to model the DTU invoice system, and supports the addition of aspects. The AspectK language takes the approach that input actions should be trapped before a concrete tuple has been selected for input, this is very useful in the LUNAR system where we are focused on checking a security policy. If we were to trap after a concrete tuple has been selected for input, it would constitute a covert channel [G99] as the presence or absence of a tuple in the tuple space might enable or prevent the advice to trap the action, and this would amount to visible behaviour bypassing the security policy. In [HNNY08] a successful method was introduced for trapping an input action before a concrete tuple has been selected for input. This has the ability to deal with joinpoints that contain constructs for binding new variables. Such joinpoints are referred to as 'open joinpoints' in [HNNY08] and we shall do the same.

At the time of checking no variable has been bound to values in the tuplespace and hence the check is not dependant upon specific values, this implies that effectively all possible bindings in the tuplespace are trapped by the aspect. In addition, AspectK performs the trapping and execution of an action as a single atomic step avoiding time-of-check-to-time-of-use race conditions. AspectK aspects allow a number of actions for handling a trap, including the option to to block an action or to let it proceed, which is sufficient for the LUNAR system. For these reasons we have chosen to use a slightly modified version of AspectK, into which to transform our security policy, the modifications we have introduced are very small and will be introduced as part of the description of the AspectK syntax below.

4.1 AspectK

The AspectK syntax is an extension of the subset of the KLAIM syntax shown in Table 1. AspectK aspects have two main elements, the *cut* and the *body*. The *cut* is the part that is matched against actions being executed and decides whether the aspect in question should be applied to the executing action. If the *cut* matches the action then the *body* of the aspect is executed. The *body* has to contain either a break statement which stops the action from being executed, or a proceed statement which allows the execution to proceed. The aspect body can also have actions that are executed before a break statement, or before and after a proceed statement. Variables that are bound in the action can only be accessed

$$
\begin{array}{rcl}
\Phi_f(A[cut] \triangleq body, \Gamma_A; \ell :: a) & = & \text{case } trap(cut, \ell :: a) \text{ of } \quad \begin{array}{ll} \mathsf{fail}: & \Phi_f(\Gamma_A; \ell :: a) \\ \theta: & \kappa_f^{\Gamma_A, \ell::a}(body\ \theta) \end{array} \\
\Phi_f(\varepsilon; \ell :: a) & = & \text{if } size(f) > 0 \quad \begin{array}{l} \text{then: } \mathbf{stop}(f) \\ \text{else: } \underline{a} \end{array}
\end{array}
$$

Table 4: Trapping Aspects: Step 1.

$$
\begin{array}{rl}
trap(cut, \ell :: a) = & \text{case } (cut, \ell :: a) \text{ of} \\
& \quad (\ell_s :: \mathbf{out}(\overrightarrow{\ell})@\ell_0, l_s :: \mathbf{out}(\overrightarrow{l})@l_0): \quad check(\langle \ell_s, \overrightarrow{\ell}, \ell_0 \rangle, \langle l_s, \overrightarrow{l}, l_0 \rangle) \\
& \quad (\ell_s :: \mathbf{in}(\overrightarrow{\ell^\lambda})@\ell_0, l_s :: \mathbf{in}(\overrightarrow{\ell'^\lambda})@l_0): \quad check(\langle \ell_s, \overrightarrow{\ell^\lambda}, \ell_0 \rangle, \langle l_s, \overrightarrow{\ell'^\lambda}, l_0 \rangle) \\
& \quad (\ell_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@\ell_0, l_s :: \mathbf{read}(\overrightarrow{\ell'^\lambda})@l_0): check(\langle \ell_s, \overrightarrow{\ell^\lambda}, \ell_0 \rangle, \langle l_s, \overrightarrow{\ell'^\lambda}, l_0 \rangle) \\
& \text{otherwise } \mathsf{fail}
\end{array}
$$

Table 5: Trapping Aspects: Step 2.

$$check(\langle\rangle, \langle\rangle) = id$$

$$check(\langle \ldots \rangle, \langle \ell_1'^\lambda, \cdots, \ell_n'^\lambda \rangle) = id$$

$$
\begin{array}{rl}
check(\langle \ell_1^\lambda, \ell_2^\lambda, \cdots, \ell_k^\lambda \rangle, \langle \ell_1'^\lambda, \cdots, \ell_{k+n}'^\lambda \rangle) = & \text{let } \theta = \quad \text{case } (\ell_1^\lambda, \ell_1'^\lambda) \text{ of} \\
& \qquad \begin{array}{rl} (!u, !u'): & [u'/u] \\ (?u, !u'): & [u'/u] \\ (?u, l'): & [l'/u] \\ (u, l'): & [l'/u] \\ (l, l'): & \text{if } l = l' \text{ then } id \text{ else } \mathsf{fail} \end{array} \\
& \qquad \text{otherwise } \mathsf{fail} \\
& \text{in } \theta \circ check(\langle \ell_2^\lambda, \cdots, \ell_k^\lambda \rangle, \langle \ell_2'^\lambda, \cdots, \ell_{k+n}'^\lambda \rangle)
\end{array}
$$

Table 6: Trapping Aspects: Step 3.

$$
\kappa_f^{\Gamma_A, \ell::a}(\mathbf{case}\ cond\ sbody\ ;\ body) = \text{case } B(cond) \text{ of } \quad \begin{array}{ll} \mathbf{tt}: & \kappa_f^{\Gamma_A, \ell::a}(sbody) \\ \mathbf{ff}: & \kappa_f^{\Gamma_A, \ell::a}(body) \end{array}
$$

$$
\kappa_f^{\Gamma_A, \ell::a}(sbody) = \text{case } sbody \text{ of } \quad \begin{array}{rl} as_1\ \mathbf{proceed}\ as_2: & as_1.\Phi_f(\Gamma_A; \ell :: a).as_2 \\ as\ \mathbf{break}(\mathbf{msg}): & as.\Phi_{f \cup \{\mathbf{msg}\}}(\Gamma_A; \ell :: a) \end{array}
$$

Table 7: Trapping Aspects: Step 4.

in the body after a proceed statement, since before the break or proceed statements a concrete tuple has not yet been selected, and hence the variables have not been bound. AspectK also adds a new token for ℓ^λ, the $?u$ token which matches both l and $!x$ in actions.

The trapping of actions and execution of aspects is done by the functions shown in tables 4 through 7. For a more detailed discussion of how these functions work we refer to [HNNY08], we show them here with our modifications for the sake of completeness and to discuss what modifications we have made to allow wildcards and error messages to be incorporated into AspectK.

The ϕ function in Table 4 checks each action against every aspect. It uses the *trap* function from Table 5 to check whether the aspect matches the action. The *trap* function then uses the *check* function from

The Algorithm
1) Given a pattern u::op(p_1,p_2, ..., p_n)@l
2) Create tuple t as $\langle u, p_1, p_2, \ldots p_n, l\rangle$
3) Create outer case condition $case_{outer}$
4) Create inner case condition $case_{inner}$
5) for each part p_i in t:
  if (p_i is V_1+V_2+...+V_k)
    create new or-clause $match_i$
    for each value v_j in p
      add condition '$p_i = v_j$' to $match_i$
    add $match_i$ to $case_{outer}$ condition
    add parameter p_i to aspect cut
  else if (p_i is [V_1+V_2+...+V_k])
    create new or-clause $allowed_i$
    for each value v_j in p_i
      add condition '$p_i = v_j$' to $allowed_i$
    add $allowed_i$ to $case_{inner}$
    add parameter p_i to aspect cut
  else if (p_i is *)
    add parameter ?p_i to aspect cut
  else if (p_i is ...)
    add parameter ... to aspect cut

Figure 4: Algorithm for converting LUNAR policies to Aspects.

Table 6 to determine whether the entities from the cut match the entities in the action, and produce a substitution of values for variables. If the aspect matches the action then the aspect body is executed before checking the next aspect. After checking every aspect it looks at the index f to determine whether to execute the original action or halt. If any aspect has issued the break statement, then the action is not executed and the process halts.

We have made two modifications to the original AspectK definition of these functions. First, we modified the *check* function to accommodate the ... wildcard character. Our version deviates from the original in that the number of parameters does not have to be the same in both tuples, there can be k entities in the cut but $k + n$ entities are possible in the action and happens if the ... wildcard is used. The function checks the first entity in the cut tuple against the first entity in the action tuple, and produces a substitution if they match or fails if they don't. It then recursively calls itself using the tuples with the first entity removed. If both tuples are empty the function returns indicating that the aspect did match the action. If the cut tuple has only the ... wildcard and the action tuple has n entities left, the function also returns indicating that the aspect is a match, regardless of the values of the n entities left in the action tuple.

Secondly, we changed the index f and the break statement. In the original version f could take on either the value break or proceed. In our version we allow the break statement to take a parameter, which is an error message saying why the action is not permitted. We redefine f to be a set of error messages, and in ϕ we check whether the number of elements in the set f is greater than zero, if so we halt the process and display the error messages in f to the user.

4.2 Converting LUNAR patterns to AspectK aspects

The LUNAR system can read a security policy specified in the LUNAR policy language and convert the patterns to AspectK aspects. Due to the permutation invariance of LUNAR policies we can convert each pattern independently of the other patterns. Each generated aspect consists of two conditional statements. The outer conditional checks whether the matching values (values not inside [ ]) in the pattern match their corresponding elements in the action. If they do not match then the pattern is not

a match for the action, and a **proceed** advice is given. If the pattern matches the action, then the inner conditional statement will check for each of the allowed values' parameters whether the action contains allowed values. If the action contains any value that is not in the set of allowed values, then the violation is logged to a Log location and a **break** advice is given. The * wildcard in a pattern is transformed to a $?p_n$ parameter in the AspectK cut, so it can match both location constants and binding of variables, as the p_n parameter can contain anything it is not checked. We have also added the ... token to AspectK cuts, where it has the same meaning as in the LUNAR policy language, any number of parameters with any values.

We employ the algorithm shown in Figure 4 to convert LUNAR policies to Aspects.

4.3 Case study: transformation example

The transformation shown below is of the pattern that specifies which tuples can be put into Ledger's tuplespace. Ledger represents DTU's general ledger and should only receive invoices and credit notes which are paid, and only from DTUFinance.

```
"4.  Ledger allowed content"
[DTUFinance]::out([INVOICE+CREDITNOTE], [PAID], ...)@Ledger
```

is transformed into

$$
\begin{aligned}
\mathsf{A}[user :: \mathbf{out}(p1, p2, ...)@loc] \quad \triangleq \quad & \mathbf{case}(loc = Ledger) \\
& \quad \mathbf{case}(p1 = INVOICE \vee p1 = CREDITNOTE) \\
& \quad \wedge (p2 = PAID) \wedge (user = DTUFinance) \\
& \quad\quad \mathbf{proceed} \\
& \quad ; \\
& \quad\quad \mathbf{out}(Violation, user, p1, p2, loc)@Log \\
& \quad\quad \mathbf{break}\ \text{"Ledger allowed content"} \\
& ; \\
& \quad \mathbf{proceed}
\end{aligned}
$$

For **out** actions this transformation is enough and results in one aspect for each pattern. For **in** and **read** actions however we also have to consider that an action might try to bind a variable in a position where we have specified allowed values. Consider the case where we have the pattern

```
"C allowed reads"
*::read([A], [B], *)@C
```

Here we state that a **read** operation can only be performed at location C if the first parameter has the value A and the second parameter has the value B. The parameter values in the actual action can be checked against the allowed values if they are location constants or variables, but if they are variable bindings then they they will not match the AspectK cut. Consider the action

```
User::read(!x, !y, !z)@C
```

This will not match the generated AspectK cut A[u::**read**(p1, p2, ?p3)@l], and would allow the user to randomly bind to a tuple in C's tuplespace. In the case where allowed values are specified for a parameter we want to break if a user tries to bind a variable to that parameter. To do that we need the pattern in question to issue a **break** advice on all of the following actions:

```
U::read(AA, BB, !z)@C
U::read(!x, B, !z)@C
U::read(A, !y, !z)@C
U::read(!x, !y, !z)@C
```

The first action can be checked with an aspect generated from the transformation described above. For the three remaining aspects which all contain variable bindings, we generate one aspect for each possibility. These aspects only check whether the matching values in the pattern (in this case only C) match the action and then immediately break. The three extra aspects generated in this case would be

$$\mathsf{A}[user :: \mathbf{out}(!p1, p2, ?p3)@loc] \triangleq \mathbf{case}(loc = C) \; \mathbf{out}(Violation, u, p2, l)@Log \; \mathbf{break} \text{ "C Allowed reads"} \; ; \; \mathbf{proceed}$$

$$\mathsf{A}[user :: \mathbf{out}(p1, !p2, ?p3)@loc] \triangleq \mathbf{case}(loc = C) \; \mathbf{out}(Violation, user, p1, loc)@Log \; \mathbf{break} \text{ "C Allowed reads"} \; ; \; \mathbf{proceed}$$

$$\mathsf{A}[user :: \mathbf{out}(!p1, !p2, ?p3)@loc] \triangleq \mathbf{case}(loc = C) \; \mathbf{out}(Violation, user, loc)@Log \; \mathbf{break} \text{ "C Allowed reads"} \; ; \; \mathbf{proceed}$$

It is possible in AspectK to define a cut which matches all these possibilities, by having all parameters take the form $?p$, since parameters prefixed with ? match both location constants and variable bindings. So a cut such as A[u::**read**(?p1, ?p2, ?p3)@l] would indeed match any possible combination of variable bindings and location constants. The problem with this approach is that in the body of the aspect we have no way of determining whether $?p1$ and $?p1$ are variable binding or not. Because of that we cannot check their values since if they are in fact variable bindings they will have undefined values and the variables $p1$ and $p2$ should not be used in the aspect body until after a **proceed** statement.

5 Static Analysis

We employ static analysis, inspired by [NGHNNPP08] and [RPN08], to determine a reduced set of aspects needed to enforce a given security policy. The static analysis we perform is a standard abstraction of the collecting semantics of KLAIM. It works by executing the following algorithm on a KLAIM code file and an associated LUNAR policy file.

We define the following global variables for the algorithm; T_l the set of all tuples that can reside in the tuple space of a given location l. For each location's (l) processes' (p) we define σ_{lp} to denote the set of values that each variable in the process can take. Further we define the function $comb(\sigma, \overrightarrow{\ell})$ to return all possible variations of $\overrightarrow{\ell}$ by using all possible combinations of variable substitutions from σ.

Static Analysis Algorithm
1) For each location l
   Initialise T_l with the existing tuples at l.
2) Define $T' := T$ and $\sigma' := \sigma$.
3) For each locations' l processes' p action a
   if $a = out(\overrightarrow{\ell})@l_{out}$
   $T_{l_{out}} := T_{l_{out}} + comb(\sigma_{lp}, \overrightarrow{\ell})$
   if $a = in/read(\overrightarrow{\ell})@l_{in}$
   Get possible matches from $T_{l_{in}}$
   Update σ_{lp} for all $!x$ in $\overrightarrow{\ell}$
4) If $T' \neq T$ or $\sigma' \neq \sigma$ goto 3.

This algorithm works by letting the tuples at each location propagate to all the locations that they can possibly reach limited by the actions at each location. The algorithm halts as the possible values of all tuples in KLAIM are finite sets and as such the sets T and σ are finite, and eventually the stabilisation condition in step 4 will be satisfied.

To apply the analysis we define K_{lpa} which denotes the possible values for a given locations' processes' actions' variables as specified by LUNAR policies. Further we define the function $match(\alpha, K_{lpa})$ which

returns true if the lunar pattern α matches K_{lpa}. Finally, we define A to be the set of patterns that need to be implemented as aspects. We then employ the following algorithm:

Applying the analysis
1) For each locations' l processes p action a
    Use σ_{lp} to generate all possible variations of a and add them to the set K_{lpa}
2) For each lunar pattern α
    For each location's l processes p action a
     if $match(\alpha, K_{lpa})$ then
      $A := A + \alpha$

This algorithm compares the possible parameter values that can occur with the ones that are permissible given a specific security policy. The test in step 2 is the key part and allows the set A to be filled with policies that may be violated.

6 Related work

From its inception aspect oriented programming has dealt with issues of security [VBC01] such as logging, error handling and access control, by allowing you to seperate and centralize these security concerns from the main codebase. The paper [XG05] presents an aspect-oriented approach to access control in mobile agent systems, which despite being based around AspectJ is a good example of where the concept illustrated by LUNAR of using static analysis to reduce the number of aspects that actually need implementation could be beneficial. Similarly related work by the same authors [XGK06] and other work such as [WVD01] could benefit from the same approach.

The original KLAIM language [NFP98] is orientated towards mobile code and while our work is focused specifically on using aspects to enforce security policies other approaches to this problem exist. Schneider presents a formalism for modelling security policy enforcement mechanisms using a class of automata [S00]. This approach, the use of reference monitors, is somewhat dated but still the most common method of enforcing security policies on mobile code. However, other approaches have emerged which seem promising. These include proof-carrying code [N97] where untrusted code has an attached proof that the code does not violate a given security policy. The attached proof can then be checked upon execution of the code, and halted if it does not comply. A similar approach is model-carrying code [SRRS01] which extends proof-carrying code to add runtime monitoring in the cases where the proof cannot be satisfied.

7 Conclusion

We have presented the LUNAR system, and the LUNAR policy language for specifying security policies for KLAIM nets. This system has been implemented, tested and source code has been made available. This is a prototype system for adding security policies to KLAIM nets and serves as a proof of concept of a method for determining the elements of a security policy that are necessary to implement as aspects.

Given the power of aspect oriented programming, if a programmer makes a mistake when writing an aspect it can lead to widespread program failure. Therefore, reducing the set of aspects that need to be implemented should help shrink the chance of one containing a mistake. The LUNAR system serves as a proof of concept that this can indeed be done and we believe has been presented in general enough terms to be applied to other languages.

7.1 Future work

While the LUNAR system operates on KLAIM code, in future work we would like to extend the system to allow the application of a reduced set of aspects to other code, such as Java code with aspects implemented as AspectJ aspects [KLMMLLI97] and analysis done as in [ACHKLLMSST05]. In addition we would be interested in conducting a larger case study with a more diverse set of security requirements.

A Source code

This report, source code for the LUNAR system and pre-compiled Windows binaries are available from `http://www.student.dtu.dk/\~{}lthhe/`.

References

[ACHKLLMSST05] P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, J. Lhotak, O. Lhotak, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. Optimising AspectJ. *ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM Press*, 2005.

[G99] Dieter Gollmann. Computer Security. *John Wiley & Sons; 1st edition*, 1999.

[H05] Michael Havey. Essential Business Process Modeling. *O'Reilly Media; 1st edition*, 2005.

[HNNY08] Chris Hankin, Flemming Nielson, Hanne Riis Nielson and Fan Yang. Advice for Coordination. *Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, 2008. Proceedings*, 2008.

[KLMMLLI97] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. *Proceedings European Conference on Object-Oriented Programming, pages 220 242*, 1997.

[LP97] H. Lewis and Christos Papadimitriou. Elements of the Theory of Computation (2nd ed). *Prentice-Hall*, 1997.

[MK03] H. Masuhara and K. Kawauchi. Dataflow Pointcut in Aspect-Oriented Programming. *Programming Languages and Systems: First Asian Symposium*, 2003.

[N97] George C. Necula. Proof-carrying code. *Proceedings 24th Annual Symposium on Principles of Programming Languages, (Paris, France, Jan. 1997) ACM, New York, 106 - 119*, 1997.

[NFP98] R. De Nicola, G. Ferrari and R. Pugliese. Klaim: a Kernel Language for Agents Interaction and Mobility. *Transactions on Software Engineering, 24(5):315330*, 1998.

[NGHNNPP08] Rocco De Nicola, Daniele Gorla, Ren Rydhof Hansen, Flemming Nielson, Hanne Riis Nielson, Christian W. Probst, and Rosario Pugliese. From Flow Logic to Static Type Systems for Coordination Languages. *Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, 2008. Proceedings*, 2008.

[RPN08] Ren Rydhof Hansen, Christian W. Probst and Flemming Nielson. Sandboxing in myKlaim. *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria*, 2006.

[S00] F.B. Schneider. Enforcable Security Policies. *ACM Transactions on Information and System Security, pages 30-50, 2000*, 2000.

[SRRS01] R. Sekar, C. R. Ramakrishnan, I.V. Ramakrishnan and S.A. Smolka. Model-Carrying Code. *Proceedings of the 2001 Workshop on New Security Paradigms, NSPW'01, pages 23-30, New York, USA, ACM Press.*, 2001.

[VBC01] John Viega, J. T. Bloch and Pravir Ch. Applying Aspect-Oriented Programming to Security. *Cutter IT Journal, vol 14, pp.31-39*, 2001.

[WVD01] Bart De Win, Bart Vanhaute and Bart De Decker. Security through aspect-oriented programming. *Advances in Network and Distributed Systems Security, volume 206 of IFIP Conf. Proc, Kluwer Academic Publishers*, 2001.

[XG05] Dianxiang Xu and Vivek Goel. An aspect-oriented approach to mobile agent access control. *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on , vol.1, no., pp. 668-673 Vol. 1, 4-6 April 2005*, 2005.

[XGK06] Dianxiang Xu, Vivek Goel and K. Nygard. An Aspect-Oriented Approach to Security Requirements Analysis. *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International , vol.2, no., pp.79-82, Sept. 2006*, 2006.

Analysis of the Anonymity Set of Chaumian Mixes

Vinh Pham
University of Siegen, Germany

Abstract

Chaumian Mixes are systems, which provide sender and relationship anonymity, such that a third person observing the system cannot track a message from a sender to the receiver. Nevertheless, if the senders and the receivers are observed over several rounds, then only particular sets of receivers are reasonable contact partners of a designated sender. This has been proved by intersection attacks, like the Disclosure and the Hitting Set attack [3, 5]. In these algorithms, the set of reasonable contacts of a sender is called the anonymity set. Thereby simulations have shown that the anonymity set decreases in general with the increasing number of observation rounds, such that the real communication partners of a sender can finally be revealed [1]. Our contribution is to provide analytical tools to answer the questions: "what is the bound of the anonymity set and how many rounds of observation are needed to identify the contacts of a sender?" The solution to these questions is fundamental to quantify the anonymity provided by the Mix system.

1 Introduction

Today more and more services and applications are provided to the user through networks like the Internet. The usage of these services is afflicted with the risk of losing privacy as most network protocols (e.g. TCP/IP) are not designed to provide anonymity. A well known basic concept to provide *sender-* and *relationship-anonymity* [7] is the Chaumian Mix approach proposed by Chaum in [1]. As long as more than one user sends messages through this system at a round, the receiver of a message of that single round is not linkable to its sender. Nevertheless Mixes cannot protect the persistent relationship of a sender to its contacts, from attackers who are able to observe the participating senders and receivers over a certain amount of rounds. The Disclosure and the Hitting Set attack [3, 5] show that such an adversary can draw a set of possible contacts of a designated sender, which fit to his observations. We will call this set the *anonymity set*. Our research interest is to determine and model the mentioned amount of time until the Mix network fails to protect the relation between the senders and the receivers. With this knowledge users and designers of anonymity networks can be aware of the system's vulnerability in order to invoke interventions like stop communicating, or adding dummy traffic. Since a permanent injection of dummy traffic causes a high communication overhead and therefore reduces the network performance, the knowledge of the time point of failure of the Mix system would be a mean to make dummy traffic affordable by using it only at critical time points.

The cardinality of the anonymity set is one metric for the degree of anonymity provided by the Mix system. An anonymity set of size one clearly means that there is no anonymity, as only one set of receivers can be contacted by a designated sender, while a large anonymity set means that there is a high uncertainty about the right contacts of a sender. Therefore our interest is to provide a tight analytical upper bound on the size of the anonymity set, with respect to the parameters of the Mix system (which will be specified in the next section). This bound results from the attacker's knowledge of the Hitting

[1] An exception of the general case occurs, if for example, all possible recipients are addressed by the senders at each communication round. In this case the real communication partners of a sender can not be revealed [4].

Set respectively Disclosure-attack, hence if the attacker applies one of those attacks, then the anonymity set is bounded by our bound. Since our bound is an upper bound, it does not depend on the rounds considered by the attack algorithms, hence it covers the best possible anonymity. We also prove that our bound is tight and the best possible in contrast to the known bound mentioned in [8, 5]. Although there are several algorithms in the literature, which compute the anonymity set, a tight bound is still missing. Hence our bound is not trivial.

Another anonymity metric considered in our paper is the number of observations, which the attacker has to collect to unambiguously identify the contacts of a particular sender. This is the case, where the information of the observations enables the reduction of the anonymity set to a size of one. It is of great importance for designers and users of Mix networks to know, how this metric is related to different traffic distributions. One way to conduct those analyses is the application of the Hitting Set algorithm on simulated network traffic. The drawback of this approach is the NP-completeness of the Hitting Set algorithm. Therefore we will contribute a new criterion based on discrete structures to approximate the number of rounds necessary to identify the communication partners. We will prove, that this criterion can be verified by an algorithm with only a polynomial time complexity. A former approximation approach based on discrete structures, which is comparable to our approach, is based on the 2x-exclusivity criterion suggested in [4]. However our new approach will be proved to provide a more precise approximation than 2x-exclusivity.

In the sequel we will introduce the idea of the Hitting Set attack, as well as the considered attacker and Mix model in section 2. The Hitting Set attack plays an important role throughout this paper, since our analyses are based on the knowledge of it. Our proof of the upper bound on the anonymity set will then be introduced in section 3. The proof will be derived from the complexity of our new Hitting Set algorithm, which we will also present in this section. The approximation of the number of rounds to identify the contacts of a sender will be addressed in section 4. Thereby a classification of the anonymity set will be introduced. Based on this classification we will identify one class, which is appropriate to approximate the required rounds to reveal the communication partners. Finally section 5 summarizes the main results of our work and gives a perspective to succeeding researches.

2 Hitting Set Attack

2.1 Anonymity Model

The analyses in this paper refers to a simplified model of the Mix technique, which can be generalized to other Mix systems. The users of this system are represented by a set $\mathcal{U}$ of N users. Each user $u \in \mathcal{U}$ could participate in the anonymity system as a sender or as a receiver of messages as illustrated in figure 1. The system is assumed to be round based. At each round at most b senders transmit messages to the Mix. This set of messages is denoted by the term *batch*. Our analysis focuses on the limits of the Mix system itself and not on implementation weaknesses, therefore a perfectly implemented Mix is assumed, which provides strong encryption and prevention against flooding and replay attacks. In this model a *global passive adversary* is considered, who is only able to observe the senders of a batch as well as the recipients of the messages in the batch. Note that the attacker cannot link the sender and the receiver of a particular message in a single round. Since the Mix is considered to be perfect, the adversary cannot gain any further information apart from the observations.

As the attacker is supposed to be omnipresent, he can take note of the participating senders and receivers of each round. He can particularly analyse those batches, where a designated user Alice participates as a sender. For simplicity Alice is assumed to have a constant set of m peer partners, which is called the *victim peer set* $\mathcal{H}_A$. A further simplification of this model is the assumption, that the size m of the victim

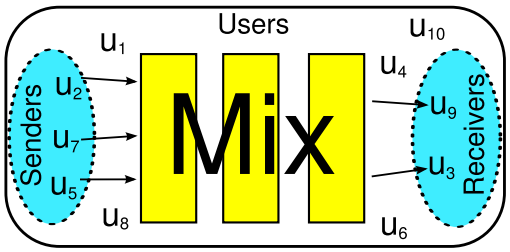


Figure 1: Subsets of users participating as senders and as receivers in a communication round of a Mix network.

peer set is known to the attacker. This assumption is without loss of generality, since an attacker could start with a small value m' and repeatedly increase its value, if there is no solution with the actual m' [2]. In order to reveal the victim peer set, the attacker only considers rounds and receivers of batches, where Alice participates. We denote those receiver sets by the term *observation* $\mathcal{O}$. Hence any observation $\mathcal{O}_i$ at the i-th round fulfills the condition $\mathcal{O}_i \cap \mathcal{H}_A \neq \emptyset$. This attacker and Mix models have been commonly used in several papers [2, 3, 5, 6, 4].

2.2 Minimal Hitting Set

Under the given anonymity model the problem of identifying the victim peer set $\mathcal{H}_A$ (which is unknown to the attacker) can be associated with the unique minimal Hitting Set problem as shown in [5]. A *Hitting Set* is a set, which intersects with all given observations $\mathcal{O}_1, \ldots, \mathcal{O}_i$. Thereby the set $\{\mathcal{O}_1, \ldots, \mathcal{O}_i\}$ is also called the *observation set*. A Hitting Set is denoted by the term *minimal*, if no proper subset of this set is a Hitting Set and it is *unique minimal*, if there is no other minimal Hitting Set of the same cardinality. Note that due to our definition, each observation contains a peer of Alice. Hence a victim peer set is a Hitting Set with respect to the observations, as it intersects with each observation. Since the attacker only considers observations, where Alice participates, the victim peer set becomes a minimal Hitting Set, if the observation set $\mathcal{OS}$ is sufficiently large. Hence to determine the unknown set $\mathcal{H}_A$ or a subset of it, it is sufficient to consider minimal Hitting Sets of cardinality at most m with respect to $\mathcal{OS}$, since each of them constitute a *hypothesis* for the full victim peer set or a subset of it. We call the set of all hypotheses with respect to $\mathcal{OS}$ the *anonymity set*.

The basic idea of the Disclosure and the Hitting Set attack [3, 5] is to determine the anonymity set with respect to a given set of observations, which is NP hard. Thereby the anonymity set is repeatedly computed by the attack algorithms for an increasing number of observations until an anonymity set is found, which consists of only one set, namely the victim peer set. In this case the attacks succeeded to identify the victim peer set, which implies that the victim peer set is a minimal and unique Hitting Set with respect to the given observations. Hence the Disclosure and Hitting Set attack empirically show us, that the anonymity set decreases with the increasing number of considered observations after reaching a threshold value on the number of hypothesis, whose bound we will prove in section 3. Finally the anonymity set converges to the set consisting of only the victim peer set. Both, the Disclosure and the Hitting Set attack compute minimal Hitting Sets. The Disclosure algorithm thereby needs at least as much or more observations to identify the victim peer set than the latter one, since it is only able to compute the anonymity set, if an observation appears, which is disjoint to $m-1$ of the previously collected m disjoint observations [3]. In contrast to this the Hitting Set algorithm can always determine the anonymity set and is therefore able to detect the uniqueness of the victim peer $\mathcal{H}_A$ on time. It was also

[2] The reader will see from the complexity analysis of section 3.3, that our attack would be changed only by a constant factor, if m is unknown.

proved to be the most efficient algorithm in [4] by requiring the least number of observations $\mathcal{O}_1, \ldots, \mathcal{O}_i$ such that $\mathcal{H}_A$ can be identified as the unique minimal Hitting Set of cardinality m. Note that efficiency refers here to the number of observations required to reveal the victim peers and not to the complexities of the algorithm. We consider in this paper only algorithms, which compute the same anonymity set for any given observation and provide the same efficiency as the Hitting Set algorithm.

It is clear that the victim peers cannot be identified, as long as the cardinality of the anonymity set is larger than one. Furthermore the cardinality of the anonymity set is a metric for the anonymity provided by the Mix system, since the effort (i.e. the number of observations) to reduce a large anonymity set is in general higher than the effort to reduce a small anonymity set. Therefore this paper will prove a strict bound on the cardinality of the anonymity set with respect to the parameters (N, m, b) of the Mix system. Additionally we will refine our consideration on the anonymity set by categorizing the Hitting Sets of the anonymity set in disjoint structures. For each of this Hitting Set structure, we will determine the upper bound of its cardinality. This is a step forward to obtain a fine grained analysis of the anonymity set. In the future work we will analyse the stochastic properties of each of the identified structures in order to mathematically model the reduction of the anonymity set with respect to increasing number of collected observations.

The following claims 1 and 2 describe, how the Hitting- respectively non Hitting-Set property of a set $\mathcal{H}$ with respect to an observation set $\mathcal{OS}$ can be transferred to particular construction of subsets of $\mathcal{H}$ and $\mathcal{OS}$. They provide the theoretical foundation for our new Hitting Set algorithm, which will be introduced in section 3. The first claim constitutes that for any Hitting Set $\mathcal{H}$ of an observation set $\mathcal{OS}$, each subset $\mathcal{H}' \subset \mathcal{H}$ is a Hitting Set of the observation set resulting from removing all observations containing any elements of $\mathcal{H} \setminus \mathcal{H}'$.

Claim 1. *Let $\mathcal{H}$ be a Hitting Set with respect to the observation set $\mathcal{OS}$. For any peer set $\mathcal{S} \subset \mathcal{H}$, the set $\mathcal{H}' = \mathcal{H} \setminus \mathcal{S}$ is a Hitting Set of the observation set $\mathcal{OS}' = \mathcal{OS} \setminus \{\mathcal{O} \in \mathcal{OS} \mid \mathcal{O} \cap \mathcal{S} \neq \emptyset\}$. If $\mathcal{H}$ is even a minimal Hitting Set then $\mathcal{H}' = \mathcal{H} \setminus \mathcal{S}$ is also a minimal Hitting Set.*

Proof. The two assertions in claim 1 will be proved by contradictions. Let $\mathcal{H}$ be a Hitting Set with respect to the observation set $\mathcal{OS}$.

- $\mathcal{H}'$ is a Hitting Set: Assume that a non empty set $\mathcal{S} \subset \mathcal{H}$ exists, so that $\mathcal{H}' = \mathcal{H} \setminus \mathcal{S}$ is no Hitting Set of the observation set $\mathcal{OS}' = \mathcal{OS} \setminus \{\mathcal{O} \in \mathcal{OS} \mid \mathcal{O} \cap \mathcal{S} \neq \emptyset\}$, which excludes observations intersecting with $\mathcal{S}$.
  Under this assumption there is an observation $\mathcal{O}_i \in \mathcal{OS}'$, not intersecting with $\mathcal{H}'$, i.e. $\mathcal{H}' \cap \mathcal{O}_i = \emptyset$. Due to the definition of $\mathcal{OS}'$, peers of $\mathcal{S}$ are not contained in any of the observations $\mathcal{O} \in \mathcal{OS}'$ either. Hence $\mathcal{H} \cap \mathcal{O}_i = \emptyset$ contradicts the assumption that $\mathcal{H}$ is a Hitting Set. Therefore $\mathcal{H}'$ must be a Hitting Set in $\mathcal{OS}'$.

- $\mathcal{H}'$ is minimal given $\mathcal{H}$ is minimal: Assume that $\mathcal{H}'' \subset (\mathcal{H}' = \mathcal{H} \setminus \mathcal{S})$ is a minimal Hitting Set of $\mathcal{OS}' = \mathcal{OS} \setminus \{\mathcal{O} \in \mathcal{OS} \mid \mathcal{O} \cap \mathcal{S} \neq \emptyset\}$.
  As for all observations $\mathcal{O} \in \mathcal{OS} \setminus \mathcal{OS}'$, the intersection $\mathcal{S} \cap \mathcal{O} \neq \emptyset$ is not empty, it follows that $\mathcal{S} \cup \mathcal{H}'' \subset \mathcal{H}$ is a Hitting Set of $\mathcal{OS}$ with a cardinality lower than $|\mathcal{H}|$. This contradicts the assumption that $\mathcal{H}$ is a minimal Hitting Set in $\mathcal{OS}$, thus $\mathcal{H}'$ must be a minimal Hitting Set in $\mathcal{OS}'$.

□

Analogous to claim 1, we also consider the case that $\mathcal{H}$ is no Hitting Set with respect to a given $\mathcal{OS}$. It will be shown that any subset $\mathcal{H}' \subset \mathcal{H}$ is no Hitting Set of the observation set resulting from removing observations containing any elements of $\mathcal{H} \setminus \mathcal{H}'$ from $\mathcal{OS}$.

Claim 2. *Let $\mathcal{H}$ be no Hitting Set with respect to the observation set $\mathcal{OS}$. For any peer set $\mathcal{S} \subset \mathcal{H}$, the set $\mathcal{H}' = \mathcal{H} \setminus \mathcal{S}$ is no Hitting Set of the observation set $\mathcal{OS}' = \mathcal{OS} \setminus \{\mathcal{O} \in \mathcal{OS} \mid \mathcal{O} \cap \mathcal{S} \neq \emptyset\}$.*

Proof. Let $\mathcal{H}$ be any set, which is no Hitting set with respect to the observation set $\mathcal{OS}$. Hence there exists at least one observation $\mathcal{O} \in \mathcal{OS}$, such that $\mathcal{H} \cap \mathcal{O} = \emptyset$. For any subset $\mathcal{S} \subset \mathcal{H}$, the set $\mathcal{OS}' = \mathcal{OS} \setminus \{\mathcal{O} \in \mathcal{OS} \mid \mathcal{O} \cap \mathcal{S} \neq \emptyset\}$ always contains $\mathcal{O}$. Since $\mathcal{H} \cap \mathcal{O} = \emptyset$, the intersection of any $\mathcal{H}' = \mathcal{H} \setminus \mathcal{S}$ and $\mathcal{O}$ is also empty, therefore $\mathcal{H}'$ is no Hitting Set of $\mathcal{OS}'$. □

3 Computation of the Anonymity Set

This section introduces the new Hitting Set attack algorithm "ExactH", which computes Alice's anonymity set with respect to the observation set collected by an adversary. Thereby the size of the anonymity set computed by the algorithm also mathematically determines its run-time complexity and vice versa. Hence we will prove the anonymity set bound by deriving the run time complexity of our new algorithm. The surprising result of our proof is that the anonymity set is bounded by b^m, which is independent of the number of users N of the Mix system. It should be noted that former approaches [8, 5] assume that the anonymity set is bounded by $\binom{N}{m}$. The reason for this loose latter bound of is due to the structure of the Hitting Set algorithm "HS-Attack" introduced in [5]. This former algorithm starts with an initial set of all possible $\binom{N}{m}$ recipient sets of size m and excludes those sets, which do no intersects with each observation, every time a new observation is collected by the adversary. Hence there are a lot of non minimal Hitting Sets computed by "HS-Attack", which blurs the real amount of minimal Hitting Sets. We will also prove that our upper bound of b^m is the best possible. Since the structure of the old Hitting Set algorithm does not allow us to formally determine this bound for the reason mentioned above, we therefore introduce our new algorithm called "ExactHS" in this paper.

As mentioned above, the upper bound depends of the run-time complexity of our Hitting Set algorithms. Thus in order to prove that our bound is correct, we must prove the correctness and completeness of our algorithm. In other words, it must be shown that "ExactHS" computes only Hitting Sets and then it must be proved that all minimal Hitting Sets are computed by the algorithm.

3.1 ExactHS Algorithm

Our new algorithm 1 "ExactHS" is a recursive function, which determines *candidate sets* $\mathcal{H} \subset N$ of size at most m and verifies whether they are Hitting Sets. Any Hitting Set found by the algorithm will be added to the set $allHS$. In order to determine a candidate set $\mathcal{H} = \{u_1, \ldots, u_m\}$, algorithm 1 chooses the i-th peer u_i at the i-th recursion level of "ExactHS" for $i \in \{1, \ldots, m\}$. Thereby the *first recursion level* is the first invocation of "ExactHS" and the *second recursion level* are those invocations of "ExactHS" at the first recursion level. The remaining recursion levels are defined analogously. Since the algorithm computes Hitting Set of cardinality at most m, the recursion level cannot be deeper than m.

Initially the set $\mathcal{H}$ is empty and the observation set $\mathcal{O}S$ consists of all observations collected by the attacker, hence the computation of the anonymity set is initially invoked by calling $ExactHS(\mathcal{OS}, m, \mathcal{H})$. We will address this observation set by the term *initial observation set*, as $\mathcal{O}S$ will be changed during the processing of "ExactHS". In each recursion level $\mathcal{H}$ is extended by exactly one peer u, chosen in line 7 from a *designated observation* $\mathcal{O} \in \mathcal{O}S$ determined in line 5, where $\mathcal{O}S$ is the actual observation set. At the invocation of the next recursion level to determine the next peer to be added to $\mathcal{H} \cup \{u\}$ in line 8, "ExactHS" is applied to a modified copy of the actual observation set, which contains no observations intersecting with $\{u\}$. Note that due to the claims 1 and 2 removing those observations has no influence on whether the remaining chosen peers together with $\mathcal{H} \cup \{u\}$ will be a Hitting Set (of the initial observation set) or not. This step of removing is important to restrict the run-time of our algorithm, as it allows us to focus on adding only those peers to the actual set $\mathcal{H} \cup \{u\}$, which definitively intersect with

observations not intersected by $\mathcal{H} \cup \{u\}$. Finally, if line 2 detects no observation of the actual observation set remains, which is not intersected by $\mathcal{H}$, then $\mathcal{H}$ is a Hitting Set. In this case it will be added to the set $allHS$ in line 3.

After a selection of u has been done in a recursion level, we remove u from all observations of the actual observation set in line 8 and from the designated observation $\mathcal{O}$ in line 9. On that way the algorithm can repeat the extension of $\mathcal{H}$ with a new peer u not chosen before in line 7.

Note that our algorithm computes the same minimal Hitting Set as the "HS-Attack" proposed in [5]. The difference is that our algorithm allows us to prove the upper bound. Additionally section 3.3 will show that "ExactHS" also provides a significant better space complexity than "HS-Attack". If the kind of applied algorithm ("ExactHS", "HS-Attack") is not important, then we simply talk about Hitting Set algorithm or attacks.

Algorithm 1 ExactHS

```
1:  procedure ExactHS(OS, m', H)
2:      if OS = {} then
            # H is a Hitting Set, add it to allHS
3:          allHS ← allHS ∪ {H}
        # add a peer to H, if it contains less than m peers
4:      else if m' ≥ 1 then
5:          choose O ∈ OS
6:          while (|O| > 0) ∧ ({} ∉ OS) do
                # u will become element of H
7:              choose u ∈ O
                # select remaining (m' − 1) peers of H
8:              ExactHS(OS \ {O_i ∈ OS | u ∈ O_i}, m' − 1, H ∪ {u})
                # remove u in all observations of OS
9:              OS ← ⋃_{O_i ∈ OS} {O_i \ {u}}
                # do not choose u in this recursion level again
10:             O ← O \ {u}
11:         end while
12:     end if
13: end procedure
```

Claim 3 (Anonymity set cardinality)**.** *Let $\mathcal{H}_A$ be a static victim peer set of size m. Let the number of users participating in the Mix system be N and the size of the batches be at most b. For a given observation set $\mathcal{OS}$, the maximal size of the anonymity set, i.e. the maximal possible number of minimal Hitting Sets of cardinality less or equal to m in $\mathcal{OS}$ is b^m. This bound is a sharp upper bound if $mb \leq N$.*

We will prove claim 3 in two steps. First section 3.2 will prove that our algorithm is sound and complete. Hence it will be shown that algorithm 1 computes all minimal Hitting Sets of cardinality at most m with respect to a given (initial) observation set $\mathcal{OS}$. After that, section 3.3 determines the complexity of "ExactHS" and thereby proves that this algorithm cannot compute more than b^m minimal Hitting Sets of cardinality at most m. In particular there are observation sets, such that exactly b^m minimal Hitting Set can be computed, which verify that our bound is tight.

3.2 Soundness and Completeness

The first proof in this section verifies that "ExactHS" is sound and therefore only computes Hitting Sets. This means that for a given observation set $\mathcal{OS}$ and a cardinality restriction i, the algorithm determines Hitting Sets, whose cardinality are not larger than i. The second proof confirms that all minimal Hitting

Sets are computed by "ExactHS", i.e. the set $allHS$ contains all minimal Hitting Sets at the termination of the algorithm. These two proofs shows that our algorithm is correct.

Soundness. The soundness of Algorithm 1 can be proved by contradiction. Assume that ExactHS computes a Set $\mathcal{H}'$, which is no Hitting Set. This means that there is an observation $\mathcal{O}_i \in \mathcal{OS}$, so that $\mathcal{H}' \cap \mathcal{O}_i = \emptyset$. Without loss of generality let $\mathcal{H}' = \{u_1, \ldots, u_i\}$, where u_j is the peer chosen by "ExactHS" in the j-th recursion level, for $j \in \{1, \ldots, i\}$. Furthermore we address wlog. by the j-th recursion level only the event, where u_j is chosen.

We observe that after choosing u_j in the j-th recursion level, line 8 invokes the $(j+1)$-th recursion level by submitting a copy of the actual observation set $\mathcal{OS}$, where all observations containing the peer u_j are removed. Hence in the $(i+1)$-th recursion level (i.e. the last level) the considered observation set is the initial observation set, where all observations intersecting with $\mathcal{H}'$ are removed. Therefore line 2 will find a non empty set, which contains at least $\mathcal{O}_i$ (which is in accordance to claim 2). This implies that $\mathcal{H}'$ is not accepted as a Hitting Set by the algorithm, which contradicts to the assumption that "ExactHS" would consider $\mathcal{H}'$ as a Hitting Set. Thus "ExactHS" only computes Hitting Sets and it is obvious that those sets cannot have cardinality larger than i because of the count down in line 8. □

Completeness. The completeness of algorithm 1 will be proved by induction on the cardinality of minimal Hitting Sets, i.e. it will be shown that ExactHS identifies all minimal Hitting Sets of a given cardinality.

Ind. basis: Let $\mathcal{H}_1$ be a minimal Hitting Set of cardinality 1 with respect to the observation set $\mathcal{OS}_1$. We prove that ExactHS will compute $\mathcal{H}_1$ given the parameters ($\mathcal{OS} = \mathcal{OS}_1$, $m' = 1$, $\mathcal{H} = \{\}$). As $\mathcal{H}_1$ is a minimal Hitting Set, each $\mathcal{O} \in \mathcal{OS}_1$ contains at least one peer $u \in \mathcal{H}_1$. Hence any choice of observations taken by the algorithm in line 5 will embody at least a $u \in \mathcal{H}_1$. Let $\mathcal{O}_k$ be the chosen observation in line 5 and the algorithm chooses $u \in \mathcal{O}_k \cap \mathcal{H}_1$ in line 7. Due to line 8 of algorithm 1, the resulting observation set $\mathcal{OS}'_1$ submitted to the recursive invocation of ExactHS excludes any observations $\mathcal{O}$, where $u \in \mathcal{O}$. As the minimal Hitting Set $\mathcal{H}_1$ only consists of u, all observations $\mathcal{O} \in \mathcal{OS}_1$ are excluded and $\mathcal{OS}'_1 = \mathcal{OS}_1 \setminus \{\mathcal{O} \in \mathcal{OS}_1 \mid u \in \mathcal{O}\}$ is the empty set $\{\}$, hence $\{u\}$ is a minimal Hitting Set computed by algorithm 1.

Ind. step: Assume that $\mathcal{H}_i$ is a minimal Hitting Set of cardinality $|\mathcal{H}_i| = i$ with respect to $\mathcal{OS}_i$ and that the invocation of algorithm 1 with the parameters ($\mathcal{OS} = \mathcal{OS}_i$, $m' = i$, $\mathcal{H} = \{\}$) will compute $\mathcal{H}_i$. We prove that the assumption also hold in the case of minimal Hitting Sets $\mathcal{H}_{i+1}$ of cardinality $|\mathcal{H}_{i+1}| = i + 1$ with respect to $\mathcal{OS}_{i+1}$.
With the same argumentation as in the induction basis, any choice of an observation $\mathcal{O} \in \mathcal{OS}_{i+1}$ in line 5 of ExactHS must include at least one peer $u \in \mathcal{H}_{i+1}$. Let us assume that $\mathcal{O}_k$ is chosen in line 5 and $u \in \mathcal{O}_k \cap \mathcal{H}_{i+1}$ is selected in line 7. Hence the resulting observation set $\mathcal{OS}'_{i+1} = \mathcal{OS}_{i+1} \setminus \{\mathcal{O} \in \mathcal{OS}_{i+1} \mid u \in \mathcal{O}\}$ in the recursive call of ExactHS in line 8 contains due to claim 1 $\mathcal{H}_{i+1} \setminus \{u\}$ as a minimal Hitting Set. According to the induction assumption any minimal Hitting Sets of the cardinality $i = |\mathcal{H}_{i+1} \setminus \{u\}|$ can be computed by the algorithm. Thus the recursive invocation of algorithm 1 on the parameters ($\mathcal{OS} = \mathcal{OS}'_{i+1}$, $m' = i$, $\mathcal{H} = \{u\}$) in line 8 reveals the minimal Hitting Set $\mathcal{H}_{i+1}$.

It follows from the induction steps that for all minimal Hitting Set $\mathcal{H}$ of an observation set $\mathcal{OS}$, where $|\mathcal{H}| = m$, algorithm 1 always indentify $\mathcal{H}$ given $\mathcal{OS}$ and m. As this holds for all minimal Hitting Sets of $\mathcal{OS}$ with a cardinality not greater than m, ExactHS is also complete in the sense that it computes all minimal Hitting Sets of $\mathcal{OS}$ with cardinality at most m. □

Note that the above proof does not exclude, that "ExactHS" could also compute Hitting Sets, which are not minimal. We could apply a minimality check on the Hitting Sets computed by the algorithm to

ensure that only minimal Hitting sets are added to $allHS$. But this effort is not worth the work, since it would not change the upper bound of the anonymity set as can be seen in the next section.

3.3 Complexity

Let us assume that the size of each observation is restricted by b, the cardinality of the victim peer set is m and that an observation set $\mathcal{OS}$ is given. We will determine the run-time of algorithm 1 with respect to the given parameters by first counting the number of Hitting Sets and non Hitting Sets constructed and considered by the algorithm. Remember that a set is constructed by extending the initially empty set $\mathcal{H}$ by a peer at each recursion level. Thereby an extension of $\mathcal{H}$ is terminated either by line 2, because $\mathcal{H}$ is a Hitting Set, or by line 4, because it reaches the maximal cardinality and is no Hitting Set. It can be observed, that regardless of the peers contained by the actual set $\mathcal{H}$, there are at most $|\mathcal{O}| \leq b$ possibilities to extend $\mathcal{H}$ by a $u \in \mathcal{O}$ in each recursion level in line 8. Since the depth of the recursion level is restricted to $m + 1$, we can conclude that the maximal number of sets (Hitting Sets and non Hitting Sets together) considered by "ExactHS" cannot be more than b^m. Note that the $(m + 1)$-th recursion level only verify the Hitting Set property of a set and does not determine peers to extend the set.

In order to show that our bound of b^m on the number of minimal Hitting Set is tight, we next describe simple observations, which contain exactly b^m minimal Hitting Sets of size m. Assume that m pairwise disjoint observations of size b are given, then combining these b observations with each other obviously results in b^m different minimal Hitting Sets of size m.

As a concrete example we can choose the parameter $m = 2, b = 2$ with the victim peer set $\mathcal{H}_A = \{1, 2\}$ and the observations $\{1, 3\}, \{2, 4\}$. A short glance shows that there are $b^m = 4$ minimal Hitting Sets, namely:

$$\{1, 2\}, \{1, 4\}, \{3, 2\}, \{3, 4\}$$

So far we only counted the number of sets considered by "ExactHS". To obtain the run-time complexity we account that extending $\mathcal{H}$ by a peer u requires the removal of all observations containing u from the observation set in line 8. The cost for this action is $O(tb)$, where t is the number of observations collected by the attacker. Hence choosing a set of cardinality m requires $O(mtb)$ operations. Since algorithm 1 constructs at most b^m distinct sets, it can be followed that the run-time complexity is:

$$O(b^m mtb).$$

Deriving the space complexity is also straight forward. We observe that each recursion level keeps a copy of the actual observation set. Since the maximal recursion level is restricted to $m + 1$, it can be concluded that the maximal space is bounded by:

$$O((m + 1)tb),$$

where tb is the maximal size of the observation set. Hence "ExactHS" is a linear space algorithm. Note that we do not account the size of the set $allHS$, since the Hitting Sets in $allHS$ are not needed for the processing of algorithm 1, i.e. we could replace line 3 by a simple print out. In contrast to our algorithm, the Hitting Set algorithm "HS-Attack" presented in [5] computes all $\binom{N}{m}$ possible sets of cardinality m and then removes those sets, which are no Hitting sets. Thus the space complexity of "HS-Attach" is at least as large as the number of remaining minimal Hitting Sets, which we proved to be bounded by b^m. It follows from this, that the space complexity of "HS-Attack" is at least $O(b^m)$.

4 Analysis of the Anonymity Set

The last section derives the upper bound on the anonymity set of the Mix system, i.e. the maximal number of minimal Hitting Sets determined by the Hitting Set attack, if the cardinality m of the victim

peer set is given. In order to refine the analysis on the anonymity set, we will partition the Hitting Sets of the anonymity set in disjoint classes in section 4.1. Thereby Hitting Sets will be classified according to the number of victim peers contained. Using this classification section 4.2 will show that a particular class has the property to provide a good approximation of the effort to disclose the victim peer set, while being efficiently computable. We will present simulations, which visualize that on the average, if there is no Hitting Set within this class, then the victim peer can be identified uniquely. It will also be proved that our criterion to approximate the number of observation to disclose the victim peer set by the Hitting Set approach is better than the 2x-exclusivity criterion presented in [4].

4.1 Refined Hitting Set Structure

Claim 3 can be further refined to determine the maximal number of minimal Hitting Sets of the following disjoint structures:

$$\begin{aligned}
\mathfrak{H}_0 &= \{\{x_{1_1}, x_{1_2} \ldots, x_{1_m}\}\} = \{\mathcal{H}_A\} \\
\mathfrak{H}_1 &= \{\{y_{1_1}, x_{1_2} \ldots, x_{1_m}\}\} \subseteq (U \setminus \mathcal{H}_A) \times \mathcal{H}_A^{m-1} \\
\mathfrak{H}_2 &= \{\{y_{2_1}, y_{2_2}, x_{2_3} \ldots, x_{2_m}\}\} \subseteq (U \setminus \mathcal{H}_A)^2 \times \mathcal{H}_A^{m-2} \\
&\vdots \\
\mathfrak{H}_m &= \{\{y_{m_1}, \ldots, y_{m_m}\}\} \subseteq (U \setminus \mathcal{H}_A)^m
\end{aligned}$$

In the definition of these structures, the variable y refers to non victim peers, while the variable x refers to victim peers. For any Hitting Set $\mathcal{H} \in \mathfrak{H}_i$, we require that all $x_{i_k}, x_{i_l} \in \mathcal{H}$ are disjoint, for $k, l \in \{i+1, \ldots, m\}$. This is contrary to the case of the non victim peer variables, where $y_{i_k} = y_{i_l}$ is allowed for $k \neq l$.

Intuitively the Hitting Set structure $\mathfrak{H}_i$ represents minimal Hitting Sets resulting from replacing i victim peers by non victim peers in $\mathcal{H}_A$. The Hitting Set structure $\mathfrak{H}_0$ consists only of the victim peer set. A Hitting Set $\mathcal{H}$ belongs to the structure $\mathfrak{H}_i$, if and only if $\mathcal{H} \in \mathfrak{H}_i$.

The maximal number of Hitting Sets $|\mathfrak{H}_i|$ in the structure $\mathfrak{H}_i$ can be derived by a slight modification of algorithm 1. Let us fix a particular subset $\{x_{i_{i+1}}, \ldots, x_{i_m}\} \subseteq \mathcal{H}_A$ of the victim peer set of cardinality $m - i$. In order to determine all minimal Hitting Sets of cardinality equal or less than m, whose only victim peers are exactly $\{x_{i_{i+1}} \ldots, x_{i_m}\}$ the next three steps is to be executed. Firstly all observations containing those peers are removed from the initial observation-set. Secondly all victim peers are deleted from the remaining observations, which results in the modified observation set

$$\mathcal{OS}' = \{\mathcal{O} \setminus \mathcal{H}_A \mid \mathcal{O} \in \mathcal{OS}, \mathcal{O} \cap \{x_{i_{i+1}}, \ldots, x_{i_m}\} \neq \emptyset\}.$$

Thirdly algorithm 1 is applied on $\mathcal{OS}'$ to compute all minimal Hitting Set of size at most i. Clearly joining any minimal Hitting Set of the last step with the set $\{x_{i_{i+1}} \ldots, x_{i_m}\}$ results in a Hitting Set of cardinality of at most m, which belongs to the structure $\mathfrak{H}_i$. Note that step one ensures that $\{x_{i_{i+1}} \ldots, x_{i_m}\}$ will be included in the final Hitting Sets, while step two guaranties that there will be no other victim peers in the resulting Hitting Sets.

It can be observed that there are $\binom{m}{m-i}$ possibilities to choose $m - i$ different victim peers out of the victim peer set $\mathcal{H}_\mathcal{A}$. Calling algorithm 1 in step three will compute at most $(b-1)^i$ different Hitting Sets. The base of $b - 1$ is due to step two, which removes victim peers from all observations. Hence there are at most

$$|\mathfrak{H}_i| = \binom{m}{m-i}(b-1)^i = \binom{m}{i}(b-1)^i \tag{1}$$

minimal Hitting Sets belonging to the structure $\mathfrak{H}_i$. This bound is exact, as the cumulative sum of the minimal Hitting Sets of all structures $\mathfrak{H}_i$ is b^m.

$$\sum_{i=0}^{m} \binom{m}{m-i} (b-1)^i = \sum_{i=0}^{m} \binom{m}{i} (b-1)^i 1^{(m-i)} = b^m \tag{2}$$

Figure 2 plots the cardinality of the minimal Hitting Set structures $\mathfrak{H}_i$ for $i \in \{1, \ldots, m\}$ for an anonymity system with $N = 400$ users with batch sizes b of $10, 15, 20, 25$ and $m = 20$ victim peers. Thereby the vertical axis is logarithmically scaled. One observes that the Hitting Set structure $\mathfrak{H}_1$ is always by far the smallest.

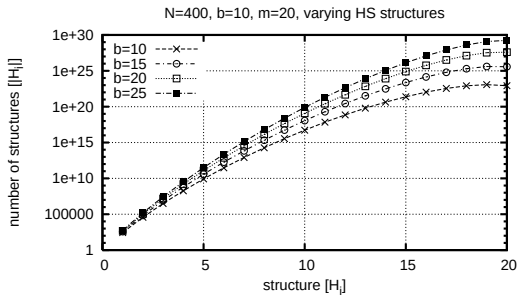


Figure 2: Maximal cardinality of different Hitting Set structures $\mathfrak{H}_i$.

4.2 Meaning of $\mathfrak{H}_1$ to the Hitting Set Attack

It is clear that, if all Hitting Set structures $\mathfrak{H}_1, \ldots, \mathfrak{H}_{m-1}$ disappear, then the victim peer set is unique. We can observe in figure 3 that in particular the structure $\mathfrak{H}_1$ is very long lasting, so that we can use the time point (i.e. the number of observations), when it disappears as an approximation for the number of observations until the victim peer set becomes unique. This is an interesting result, since the last section shows that the cardinality of this structure is only $m(b-1)$. This is by far the smallest structure apart from the victim peer set itself. Figure 3 also visualizes that $\mathfrak{H}_1$ is a better approximation of the time point of full disclosure of the victim peer set than the "2x-exclusivity" property introduced in [4].

According to [4] a victim peer appears *exclusively* in an observation, if there is no other victim peer in this observation. And a victim peer is *2x-exclusive*, if it appears at least two times exclusively in an observation, or at least one time alone (i.e. without any other peers) in an observation. The *2x-exclusivity* property is fulfilled, if all victim peers are 2x-exclusive. It was shown in [4] that this property is a good approximation for the lower bound on the number of observations to fully disclose the victim peer set. We can prove that our structure $\mathfrak{H}_1$ is a better approximation than 2x-exclusivity by showing the following two cases. First, there are observations, where the 2x-exclusivity property is fulfilled, although there is still a Hitting Set of the structure $\mathfrak{H}_1$. Second, if the 2x-exclusivity property is not fulfilled, then there is always a Hitting Set of the structure $\mathfrak{H}_1$.

Case 1. Since this is an existence proof, we only have to construct an example, where 2x-exclusivity is given, although a Hitting Set of the structure $\mathfrak{H}_1$ exists. Let the victim peer set be $\mathcal{H}_A = \{1, 3\}$. Assume that the following four observations are given:

$$\{3, 5, 6\}, \{3, 1\}, \{1\}, \{6, 3\}.$$

All victim peers in $\mathcal{H}_A$ are obviously 2x-exclusive, but there is a Hitting Set $\{1, 6\}$ of structure $\mathfrak{H}_1$. $\square$

Case 2. Let $\mathcal{H}_A = \{u_1, \ldots, u_m\}$ be the victim peer set. Assume wlog. that the victim peer u_1 appears only one time exclusively with respect to a given observation set $\mathcal{OS}$. Let $\mathcal{O} \in \mathcal{OS}$ be the observation, in which u_1 appears exclusively. Then the set $\mathcal{H} = \{y, u_2, \ldots, u_m\}$, where $y \in \mathcal{O} \setminus \{u_1\}$ is a Hitting Set of the structure $\mathfrak{H}_1$. □

Note that the structure $\mathfrak{H}_1$ is efficiently computable by the modification of algorithm 1 suggested in section 4.1. The run-time complexity to compute Hitting Sets from this structure is $O(m(b-1)mtb)$, which is polynomial.

The plots from figure 3 represents results of our simulation of the Hitting Set attack on a simulated Mix traffic. In this simulation we assume that in each round, Alice chooses her receiver uniformly distributed from one of her m peers, i.e. with probability $\frac{1}{m}$. All the other $b-1$ senders are assumed to choose their peers uniformly from the set of all user N, i.e. with probability $\frac{1}{N}$.

Each of the figures varies one of the parameter N, m, b, while all the other parameters remain fixed. The standard parameter of this simulation is $N = 400$, $m = 10$, $b = 10$. The line labelled (HS) shows the mean number of observations of the Hitting Set attack to reveal the victim peer set. Accordingly the line labelled (2x-excl) is the mean number of observation to fulfil the 2x-exclusivity criterion, while the line labelled (h_1) visualizes the mean number of observations until the structure $\mathfrak{H}_1$ disappears. Note that we assume the uniform distribution only for ease of our simulation. We could have chosen a different (e.g. zipf-distribution) and would obtain similar results.

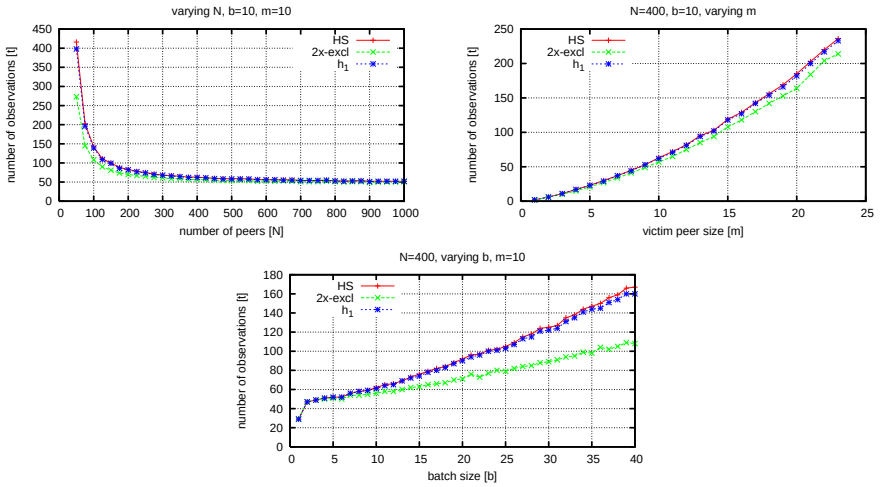


Figure 3: Mean number of observations until: uniqueness of victim peer set (HS), 2x-exclusivity property (2x-excl) and disappearance of $\mathfrak{H}_1$ structure (h_1).

5 Conclusion

The focus of this paper was on tools to quantify the anonymity provided by the Mix system. Thereby two aspects of the system were considered, the upper bound of the anonymity set and the effort (i.e. number of observation) to disclose the victim peer set.

We could prove that the bound of the anonymity set is b^m by counting the number of minimal Hitting Sets by our new algorithm "ExactHS". As a side result we also presented a proof sketch, showing that the space complexity of "HS-Attack" [5] is at least $O(b^m)$, which is open before. Thereby the space complexity of our algorithm is superior to "HS-Attack", as it is only polynomial. Also the time

complexity of our algorithm, which we proved to be $O(b^m mtb)$ is better than the runtime complexity of the "HS-Attack" algorithm.

Finally a classification of the anonymity set was introduced. Thereby a Hitting Set structure $\mathfrak{H}_1$ was presented, which provides a better approximation criterion for the effort to identify the victim peer set than the existing 2x-exclusivity criterion suggested in [4]. Our criterion is like 2x-exclusivity polynomial time computable and is therefore a practical alternative to the computation of the effort by a Hitting Set algorithm, since those algorithms are NP hard.

In the future we plan to investigate the distribution of different Hitting Set structures $\mathfrak{H}_1, \ldots, \mathfrak{H}_m$ in an anonymity set with respect to different distributions of communication partners. We have seen by the simulation that the structure $\mathfrak{H}_1$ lasts very long although it has a very small cardinality. Hence Hitting Sets in this structure are less likely to disappear. It would therefore be interesting to know the probability of finding a Hitting Set of a particular structure with respect to the number of observations to obtain a more fine grained evaluation of anonymity.

References

[1] D. L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84 – 88, February 1981.

[2] G. Danezis. Statistical Disclosure Attacks: Traffic Confirmation in Open Environments. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421 – 426, Athens, May 2003. IFIP TC11, Kluwer.

[3] D. Kesdogan, D. Agrawal, and S. Penz. Limits of Anonymity in Open Environments. In F. Petitcolas, editor, *Information Hiding: 5th International Workshop (IH2002)*, volume 2578 of *LNCS*, pages 53 – 69. Springer-Verlag, October 2002.

[4] D. Kesdogan, D. Agrawal, V. Pham, and D. Rauterbach. Fundamental Limits on the Anonymity Provided by the Mix Technique. *IEEE Symposium on Security and Privacy*, May 2006.

[5] D. Kesdogan and L. Pimenidis. The Hitting Set Attack on Anonymity Protocols. In J. Fridrich, editor, *Information Hiding: 6th International Workshop (IH2004)*, volume 3200 of *LNCS*, page 326. Springer-Verlag, May 2004.

[6] D. Kesdogan and L. Pimenidis. The Lower Bound of Attacks on Anonymity Systems – A Unicity Distance Approach. In *First Workshop on Quality of Protection*, September 2005.

[7] A. Pfitzmann and M. Hansen. Anonymity, Unobservability, Pseudonymity, and Identity Management – A Proposal for Terminology. `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml`, 2008.

[8] L. Pimenidis. Structure and Analysis of Chaumian Mixes (Struktur und Analyse von Chaummixen). Master's thesis, RWTH-Aachen, November 2003.

ZigBee-2007 Security Essentials

Ender Yüksel Hanne Riis Nielson Flemming Nielson
Informatics and Mathematical Modelling, Technical University of Denmark
{ey,riis,nielson}@imm.dtu.dk

Abstract

ZigBee is a fairly new but promising standard for wireless networks due to its low resource requirements. As in other wireless network standards, security is an important issue and each new version of the ZigBee Specification enhances the level of the ZigBee security.

In this paper, we present the security essentials of the latest ZigBee Specification, *ZigBee-2007*. We explain the key concepts, protocols, and computations. In addition, we formulate the protocols using standard protocol narrations. Finally, we identify the key challenges to be considered for consolidating ZigBee.

1 Introduction

Low-cost and low-power wireless networks have an enormous potential for many areas such as health, security, telecom, agriculture, energy, etc. ZigBee is an emerging standard for *low-rate* (in terms of cost, power consumption, range, data) wireless networks and the use of the ZigBee technology is limited only by imagination [33, 43]. However, wireless networks are generally susceptible to attacks and security is clearly very important. As a low-rate standard, ZigBee has very limited resources (memory, processor, storage, power, etc.), therefore implementing security guarantees is a great challenge and the security specification is of paramount importance.

The ZigBee Specification is difficult to read with hundreds of pages and refers to many other standards. In this study, we present a high level self-contained overview and explain the key components of the latest ZigBee Specification, *ZigBee-2007*. We explain the important points in key protocols, algorithms, and computations. We derive the protocol narrations from the specification, which will be useful for both implementations and security analyses. In addition, we survey the key studies related to ZigBee security. Therefore, this paper goes beyond a mere survey, aiming to facilitate the use and study of ZigBee in both industry and academia.

The rest of this paper is organized as follows. We present the structure and the development of the *ZigBee Specification* in Section 2. We give an overview of the *ZigBee Stack Architecture* in Section 3. We explain the *ZigBee Security Services Specification*, the core part of the ZigBee security, in Section 4. We explain the *ZigBee Security Procedures* in Section 5. We discuss the literature on the *ZigBee Security*, and present latest related work in Section 6. We conclude in Section 7, and leave the *protocol narrations* to the appendix.

2 The ZigBee Specification

ZigBee is a very low-cost, very low-power-consumption, two-way, wireless communication standard that is being developed by the *ZigBee Alliance*, an independent nonprofit organization. ZigBee is targeted at radio frequency (**RF**) applications that require low data rate, long battery life, and secure networking. ZigBee has a very wide application area that covers consumer electronics, home and building automation, industrial controls, PC peripherals, medical sensor applications, toys and games, etc.

The ZigBee Specification provides a definitive description of the ZigBee protocol standard as a basis for the implementations. The specification is a kind of *dynamic document*, which is currently called *ZigBee-2007* [5]. The history of the specification begins with the ZigBee

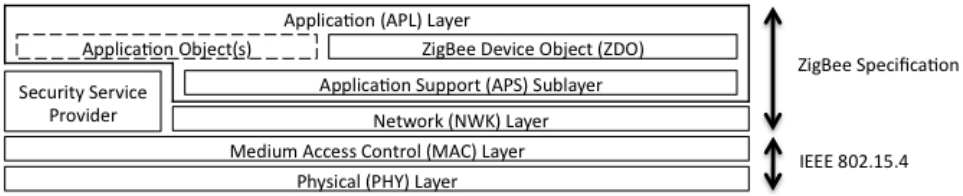


Figure 1: ZigBee Stack Architecture

v.1.0 draft that was published in December 2004 and now referred to as *ZigBee-2004* [7]. The first version of the ZigBee Specification was published in February 2006; the successor ZigBee Specification was published in October 2006 and called *ZigBee-2006* [6]. From February 2006 to present, there have been seven releases. The development is so rapid that sometimes the interval between two specifications is less than one month (*i.e. the last version of ZigBee-2006 and the first version of ZigBee-2007*). However, in the latest specification (ZigBee-2007) it is stated that there exists three main specifications: ZigBee-2004, ZigBee-2006 and ZigBee-2007. The current specification, ZigBee-2007, is required to be backward compatible with ZigBee-2006 but not necessarily with ZigBee-2004. In addition, it is stated that ZigBee-2006 is not required to have backwards compatibility with ZigBee-2004.

The ZigBee-2007 Specification [5] is composed of a short protocol overview followed by specifications of the Application Layer, Network Layer, and Security Services. In addition, the specification covers necessary details that are required in ZigBee implementations.

ZigBee-2007 contains two feature sets that interoperate network-wise: *ZigBee* and *ZigBee* PRO. A *feature set* is an agreement of configuration parameters, network settings and policies. The ZigBee PRO feature set has two security modes (Standard Security, High Security), and three types of security keys (Network Key, Link Key, Master Key), whereas the ZigBee feature set has only Standard Security mode and two types of security keys (Network Key, Application Link Key). Both feature sets use symmetric encryption, the *Advanced Encryption Standard* (**AES**-128) [3], and apply authentication/encryption on Network and Application layers.

ZigBee-2007 also has different application profiles. An *application profile* is a common application level language for exchanging data in a given application domain. The ZigBee public application profiles are *Commercial Building Automation*, *Home Automation*, *Personal Home and Hospital Care*, *Smart Energy*, *Telecom Applications*, and *Wireless Sensor Applications*.

Related to the device types, we used two different classifications: hardware and logical. The hardware device type distinguishes the type of the hardware platform and it may be either *Full Function Device* (**FFD**), or *Reduced Function Device* (**RFD**) according to the IEEE 802.15.4 standard [1]. A logical device type distinguishes devices in a specific network and it may be *Coordinator*, *Router*, or *End Device* in a ZigBee network.

Throughout this paper, ZigBee refers to the ZigBee-2007 Specification unless otherwise stated. In addition, we typed the abbreviations in boldface font, where they are first defined.

3 The ZigBee Stack Architecture

ZigBee is built on the Physical layer (**PHY**) and the Medium Access Control layer (**MAC**), both defined by the IEEE 802.15.4-2003 standard [1]. The PHY layer can operate in two separate frequency ranges: lower 868(European)/915(United States, Australia, etc.) MHz and higher 2.4 GHz (worldwide). The MAC layer controls access to the radio channel using a CSMA-CA mechanism. Upon this structure, ZigBee builds the Network layer (**NWK**) and the Application layer (**APL**) which consists of the Application Support sublayer (**APS**) and the ZigBee Device Object (**ZDO**). Fig. 1 shows the ZigBee stack architecture, including the end manufacturer defined part in dashed box. This study focuses on the Security Service Provider part of the ZigBee Specification, which interacts with the NWK and APS layers.

4 The Security Service Provider

The Security Services Specification is a chapter in the ZigBee Specification [5] which specifies the security services available within the ZigBee stack. These services include methods for key establishment, key transport, frame protection and device management. As shown in Fig. 1, ZigBee provides security mechanisms for the NWK and APS layers. Each layer is responsible for securing their frames. In addition, the APS sublayer provides services for security relationship establishment and maintenance whereas ZDO manages the security policies and the configuration of ZigBee devices. Throughout this section, we present the overall security architecture and the security mechanisms in different layers.

4.1 Open Trust Model

The ZigBee security architecture depends on assumptions such as safekeeping the symmetric keys, proper implementation of the cryptographic mechanisms and the associated security policies involved. Besides, ZigBee assumes that different applications using the same radio are not logically separated (*e.g. by firewall*). Also, a device cannot verify whether cryptographic separation between different applications/layers on another device is properly implemented. Therefore, it must be assumed that separate applications using the same radio trust each other, which means that there is no cryptographic task separation. In addition, the lower layers (i.e. APS, NWK, MAC) are fully accessible to the applications. As a result, ZigBee has the notion of an *open trust model* that is derived from these assumptions. This model states that the security services only protect the interfaces between different devices, but not the interfaces between different layers nor the different applications on the same device. Here, protection means cryptographic protection, and for the separation of the interfaces between the different stack layers on the same device non-cryptographic mechanisms should be employed in the designs. As stated in the specification, this model allows the reuse of the same keying material among different layers on the same ZigBee device. The model also allows end-to-end security to be realized on a device-to-device basis instead of layer-to-layer basis.

4.2 Architectural Design Choices

In this subsection we focus on the design choices listed in the current ZigBee Specification. The main point in these choices is that any malevolent device should not be able to use the network to transport frames across the network without permission.

1. *The layer that originates a frame is responsible for initially securing it.* A simple example from the specification: If a NWK command frame needs protection, then NWK layer security must be used.

2. *If protection from theft of service is required, then NWK layer security shall be used for all frames.* Only a device that joined the network and authenticated (i.e. received the active network key) will be able to communicate using the network. However, it is not possible to apply NWK layer security between a router and a joined but unauthenticated device.

3. *Security can be based on the reuse of keys by each layer.* This reduces the storage costs.

4. *End-to-end security is enabled, based on a shared key between only two devices.* This actually limits the trust requirement to only two devices communicating.

5. *The security level used by all devices in a given network, and by all layers of a device shall be the same.* If an application requires more security, then it shall form its own separate network with a higher security level.

Other decisions such as key update/expire, counter overflow, loss of synchronization, error conditions arising from securing frames, etc. should be included in the application profiles and must be addressed correctly in the real implementations.

4.3 Security Keys and Trust Center

ZigBee devices use 128-bit symmetric encryption keys to provide security amongst a network. A *Link Key* (**LK**) is shared by two ZigBee devices and it is used to secure unicast communication

Table 1: Key Acquirement Schemes and Availability of Key Types

Keys	Key Acquirement Schemes			Availability of Key Types			
				Layers		Modes	
	Key Transport	Key Establishment	Pre-installation	NWK	APL	SS	HS
NK	YES	NO	YES	YES	YES	YES	YES
MK	YES	NO	YES	NO	YES	NO	YES (O)
LK	YES	YES	YES	NO	YES	YES (O)	YES (O)

Table 2: Derived Key Types (0x: Hexadecimal)

Key Type	Computation	Explanation
Key-Transport Key	$\text{HMAC}(0x00)_{LK}$	Protects transported NKs
Key-Load Key	$\text{HMAC}(0x02)_{LK}$	Protects transported MKs and LKs
Data Key	LK	Equal to the LK

between APL peer entities. LK is used as the basis of the security services (i.e. key transport, authentication) in *High Security* mode (**HS**). A *Network Key* (**NK**) is shared amongst all devices in a network and it is used to secure broadcast communications in a network. NK is used as the basis of the security services (i.e. authentication, frame security) in *Standard Security* mode (**SS**). A *Master Key* (**MK**) is used to establish a key and it is shared pairwise between two ZigBee devices. The intended recipient is always aware of the key type that is used in frame protection.

The security keys can be acquired in different ways depending on their types as shown in Table 1. In *Key Transport*, the Trust Center of the network sends the key to the device. *Key Establishment* is the method that is used to establish a pairwise key (*i.e. LK*) between two devices. Note that for this method, a pre-shared key (*i.e. MK*) is required between two devices. In *Pre-installation*, the device acquires the key before joining the network.

In ZigBee-2007, NKs have two different types: standard (**SNK**), and high-security (**HSNK**). It is stated that NK distribution and initialization of the Frame Counters (*See Subsection 4.4*) depend on the type of the NK, but in both cases the messages are secured in the same way. The availability of the security keys to the different layers and different security modes are given in Table 1. Some key types are optional for some security modes and they are indicated with "(O)" in the table. In addition, all ZigBee layers must share the active NK and associated incoming/outgoing frame counters.

In order to avoid reuse of keys across different security services, it is possible to derive different keys from the LK. Uncorrelated keys can be derived using a one-way function so that execution of different security protocols can be logically separated. Three types of secret keys can be derived from a LK as listed in Table 2. The derivation of these keys, except Data Key, requires computation of a *Keyed-Hashing for Message Authentication Code* (**HMAC**) [4]. All the derived keys must share the associated frame counters.

Trust Center In each secure ZigBee network, there exists a unique Trust Center (**TC**) application which is trusted by all the devices in the network. TC distributes keys as part of network and end-to-end application configuration management. In high-security (commercial) applications devices are using MK, whereas in low-security (residential) applications devices are using NK to initiate secure communication with TC. In parallel with the key acquirement schemes in Table 1, MK and NK can be obtained by either pre-installation or a kind of key transport which is called *in-band unsecured key transport*. Certainly, the latter option is not tolerable in vulnerable situations. The interaction between a ZigBee device and the TC for different purposes is given in Table 3.

As we mentioned before, there are three logical device types in a ZigBee network: *coordinator*, *router*, and *end device*. TC is not a device type, but an application. In a ZigBee network, the ZigBee Coordinator configures the security level (which can also be *unsecured*). The ZigBee

Table 3: Key Distribution

Purpose	Device receives from TC	Via
Trust Management	Initial MK or Active NK	Unsecured Key Transport
Network Management	Initial active NK and updated NKs	Secured Key Transport
Configuration	MK or LKs	Secured Key Transport

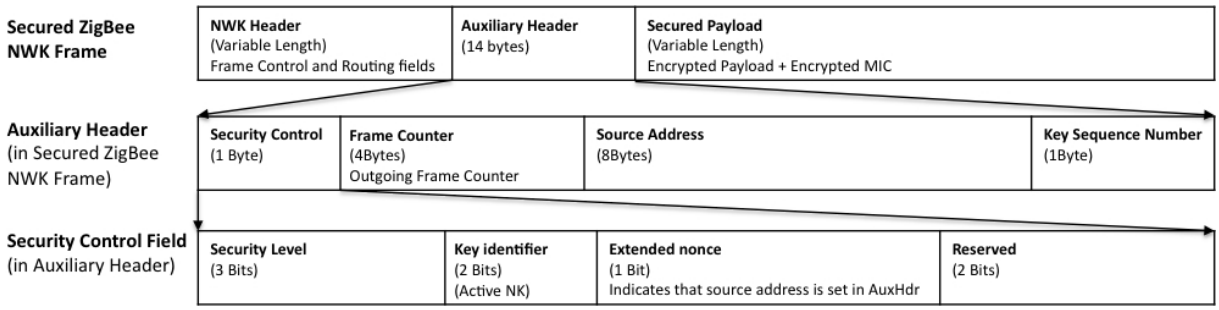


Figure 2: Secured ZigBee NWK Frame

Coordinator also configures the TC, which is by default the ZigBee Coordinator itself.

Security Modes TC can be configured to operate in either Standard Security mode (SS) or High Security mode (HS). In SS mode, which is designed for the residential applications, the TC is required to maintain the SNK and control the policies of network admittance. In HS mode, which is designed for the commercial applications, the TC is required to maintain a list of all the devices in the network (this is actually optional in SS mode), all the relevant keys (MKs, LKs, HSNKs) and control the policies of network admittance. As a result, the required memory of the TC grows with the number of the devices in the network in HS mode, but not in SS mode. In addition, the implementation of the Symmetric-Key Key Exchange (*See Subsection 4.7*) and the Mutual Entity Authentication (*See Subsection 4.8*) protocols are mandatory in HS mode. Right now, among the application profiles mentioned in Section 2, only the Commercial Building Automation (**CBA**) profile uses HS mode.

4.4 Network Layer Security

The Network (NWK) layer provides functionality to ensure correct operation of the MAC layer and also provides suitable service interface to the APL layer. When a NWK layer frame needs to be secured, the NWK layer secures it by using AES [3] encryption/authentication in the *Enhanced Counter with CBC-MAC* (**CCM***) mode of operation (*see Subsection 4.9*). The NWK layer processes outgoing/incoming frames in order to securely transmit/receive them. The upper layers control the security processing operations by setting up the security keys, frame counters and the security level.

The structure of a secured NWK frame is given in Fig. 2. A secured NWK frame is a NWK frame that has an Auxiliary Header (**AuxHdr**) and this is indicated in the *Frame Control* field of the NWK Header. AuxHdr includes the *Frame Counter*, which has the purpose of providing frame freshness and preventing processing of duplicate frames. An important point here is that the word "secured" does not necessarily mean that encryption is applied (*i.e. in the case of integrity-only protection*). Therefore, a Secured Payload does not need to have an encrypted payload. The *Security Level* field in the Security Control part of the AuxHdr indicates which security level is applied. The level can be None (ne security at all), MIC (integrity protection only, with three different MIC lengths: MIC-32, MIC-64, MIC-128), ENC (encryption only), and ENC-MIC (both encryption and integrity protection with three different MIC lengths: ENC-MIC-32, ENC-MIC-64, ENC-MIC-128). The Message Integrity Code (**MIC**) is computed using the NWK header, the AuxHdr and the (encrypted) payload.

The processing of the incoming and outgoing frames are given in Table 4. An interesting

Table 4: NWK Frame Security (||: Concatenation)

Processing Outgoing Frames	**Processing Incoming Frames**
1. Retrieve active NK, outgoing frame counter, key sequence number and security level from the NWK Layer Information Base (**NIB**). 2. Set AuxHdr using the parameters in Step 1. 3. Execute CCM* mode encryption and authentication with the parameters: MIC length (obtained from security level), NK, and CCM* Nonce (CCM* Nonce uses the values from AuxHdr and constructed as "Source Address ‖ Frame Counter ‖ Security Control". 4. Construct the outgoing frame, depending on the encryption requirement, as in Fig. 2. 5. Increase outgoing frame counter in NIB. 6. Set the *Security Level* subfield of AuxHdr as "000".	1. Determine security level from NIB and overwrite it to the *Security Level* subfield. 2. Determine the key sequence number, sender address, and the *Received Frame Count* from AuxHdr. 3. Obtain the corresponding (matching the key sequence number) security material from NIB. If *Received Frame Count* is less than *Frame Count* then FAIL. 4. Execute CCM* mode decryption and authentication with the same parameters as in outgoing frames. 5. Set Frame Count to "Received Frame Count + 1". Store Frame counter and the Sender Address in the NIB.

security precaution here is hiding the security level in the last step of Outgoing Frames Processing. Although the rationale behind this action is not defined in the specification, it is clear that it is not a significant protection since there are only eight choices (as listed above).

4.5 Application Layer Security

As shown in Fig. 1, the Application (APL) Layer is composed of the APS sublayer, the ZDO and manufacturer defined application objects. A maximum of 240 distinct application objects can be defined. ZDO is responsible for initializing APS, NWK, Security Service Provider, and assembling the information from applications. ZDOs are applications that employ NWK and APS primitives to implement ZigBee End Devices, ZigBee Routers and ZigBee Coordinators. When an APL layer frame needs to be secured, the APS sublayer handles security. Therefore, APL security actually covers APS sublayer security which is explained in Subsection 4.6.

4.6 Application Support (APS) Sublayer Security

The Application Support (APS) sublayer provides an interface between the NWK and the APL layers through a general set of services (for use of ZDO and Application Objects) provided by APS data and management entities. The APS sublayer processes outgoing/incoming frames in order to securely transmit/receive the frames and establish/manage the cryptographic keys. The upper layers issue primitives to APS sublayer to use its services. APS Layer Security includes the following services: *Establish Key*, *Transport Key*, *Update Device*, *Remove Device*, *Request Key*, *Switch Key*, *Entity Authentication*, and *Permissions Configuration Table*. Below we explain the current usage of these services, keeping in mind that they can be extended in the future.

The **Establish Key** service is the mechanism for establishing a LK between two ZigBee devices. In ZigBee-2007, Symmetric-Key Key Exchange (**SKKE**) is the method for key establishment and MK is the trust information used in key establishment (*See Subsection 4.7*). Recently, Public-Key Key Establishment (**PKKE**) is included in a ZigBee application profile (*See Subsection 6.1*).

The **Transport Key** service provides secure or unsecure means to transport a NK, LK, or MK.

The **Update Device** service provides secure means for a ZigBee Router to inform the TC that a device changed its status (*i.e. joined or left the network*).

The **Remove Device** service provides secure means for the TC to inform a ZigBee Router that one of his children should be removed.

The **Request Key** service provides secure means for a device to request the *active NK* or the *end-to-end application MK* from another device (*i.e. TC*).

The **Switch Key** service provides secure means for TC to inform a device to switch to the

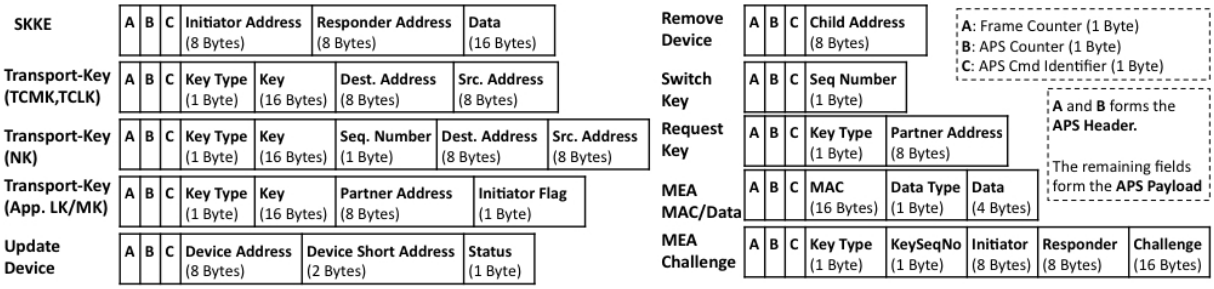


Figure 3: APS Command Frames

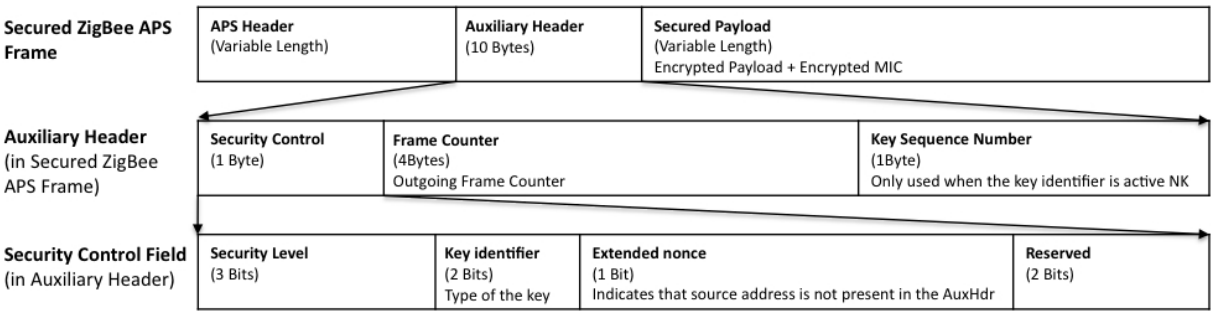


Figure 4: Secured ZigBee APS Frame

alternate NK.

The **Entity Authentication** service, *which was not present in ZigBee-2006*, provides authenticity between two devices based on a shared secret (*i.e. NK*).

The **Permission Configuration Table** (**PCT**) stores the information of which devices have authorization to perform which commands. In addition, PCT determines whether security based on LK is required or not. Maintaining a PCT is optional.

The services of the APS sublayer are issued in *APS Command Frames*. The structure of the APS Command Frames is given in Fig. 3. The first two fields of all the frames, the Frame Counter and the APS Counter, form the *APS Header*. The remaining fields form the *APS Payload*, whose first field *APS Command Identifier* indicates the type of the command frame (SKKE, Transport-Key, Update-Device, Remove-Device, Request-Key, Switch-Key, Entity Authentication, etc.), as shown in Fig. 3.

There are some important points regarding the APS command frames in Fig. 3. All *Key Establishment* (*e.g. SKKE*) command frames are sent unsecured. The *Status* field of the *Update-Device* command indicates the security mode (SS/HS), the security of the frame (Secured/Unsecured), and the type of the join (Join/Rejoin); unless the device is leaving. *Partner Address* field of the *Request-Key* command is not present when the key type is NK or TCLK (note that TCLK means LK of the TC).

We have mentioned in Table 4 that the NKs are stored in the NWK Layer Information Base (**NIB**). Likewise, the MK/LK pairs and the relevant information is stored in the APS Layer Information Base (**AIB**).

The processing of the incoming and outgoing frames in APS layer is given in Table 5. Note that the security level is hidden the same way as in the NWK layer frame security.

4.7 Symmetric-Key Key Establishment Protocol

In the Symmetric-Key Key Establishment (SKKE) protocol, an initiator device U establishes a LK (or LK_{UV}) with a responder device V using a shared secret MK (or MK_{UV}). As we mentioned in Table 1, MK may either be pre-installed or transported from the TC. We present the SKKE protocol with computational details in Table 6. In the first two messages, the devices

Table 5: APS Frame Security (||: Concatenation)

Processing Outgoing Frames	**Processing Incoming Frames**
1. Obtain the security material (from NIB or AIB), and the key identifier. If the key identifier is *active NK*, then either APS or NWK layer will apply security (not both of them). 2. Extract the outgoing frame counter (and the key sequence number if the key identifier is active NK) from the security material. 3. Obtain the security level from NIB. Set the AuxHdr (using all these parameters) as in Fig. 4. 4. Execute CCM* mode encryption and authentication, construct the CCM* nonce as "Source Address ‖ Frame Counter ‖ Security Control" (using the values from AuxHdr). 5. Construct the outgoing frame, depending on the encryption requirement, as in Fig. 4. 6. Increment and store the outgoing frame counter. 7. Set the *Security Level* subfield of AuxHdr as "000".	1. Determine the sequence number, key identifier and received frame counter value from the AuxHdr. 2. Obtain the appropriate security material from NIB or AIB depending on the key identifier. If *Received Frame Count* is less than *Frame Count* then FAIL. 3. Determine the security level from NIB and overwrite it to the *Security Level* subfield. 4. Execute CCM* mode decryption and authentication with the same parameters as in outgoing frames. 5. Unsecured APS frame will be constructed using the output of CCM*. 6. Set and store the Frame Count as "Received Frame Count + 1".

Table 6: SKKE Protocol (H: Hash Function, MAC: HMAC Function, ||: Concatenation, 0x: Hexadecimal)

		U	**V**
SKKE-1	U→V	QEU	
SKKE-2	V→U	QEV	
Computation		Z = MAC{U‖V‖QEU‖QEV}$_{MK}$ KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) MacData2 = 0x03‖U‖V‖QEU‖QEV MacTag2 = MAC(MacData2)$_{MacKey}$	Z = MAC{U‖V‖QEU‖QEV}$_{MK}$ KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) MacData1 = 0x02‖V‖U‖QEV‖QEU MacTag1 = MAC(MacData1)$_{MacKey}$
SKKE-3	U→V	MacTag2	
SKKE-4	V→U	MacTag1	
Verification		MacData2 = 0x03‖U‖V‖QEU‖QEV Verify MacTag2 using MacData2	MacData1 = 0x02‖V‖U‖QEV‖QEU Verify MacTag1 using MacData1
U and V use KeyData = H(MAC{U‖V‖QEU‖QEV}$_{MK}$‖0x00000002) as the new LK			

exchange their 16-byte challenges. In the last two messages, the devices exchange the data they have computed using the challenges and the device identities. Note that a *device identity* is the unique 64-bit device address, and ***kdf*** (defined in [2]) is the key derivation function that takes two parameters: the shared secret bit string, and the length of the keying data to be generated. After verifying that they received the correct values, they use another value as the LK that both of them can compute.

All the SKKE messages use the frame format as shown in Fig. 3. The *Data* field in a SKKE frame stores the value of either a challenge or a MAC (not the MAC layer, but the Message Authentication Code) tag.

4.8 Mutual Entity Authentication Protocol

In the Mutual Entity Authentication (**MEA**) protocol, an initiator device U and a responder device V mutually authenticate each other based on a secret key (NK). The devices authenticate each other by using random challenges with responses based on a NK. We present the MEA protocol with computational details in Table 7. In the first two messages, the devices exchange their challenges. Note that, **OFC** stands for the Outgoing Frame Counter of the device. In the last two messages, the devices exchange the data they have computed using their information and their frame counter values. They use the frame counter values they received and their

Table 7: MEA Protocol (MAC: HMAC Function, ||: Concatenation, 0x: Hexadecimal)

		U	V
MEA-1	U→V	QEU	
MEA-2	V→U	QEV	
Computation		MacData2 = 0x03‖U‖V‖QEU‖QEV‖OFC_U MacTag2 = MAC{MacData2}$_{NK}$	MacData1 = 0x02‖V‖U‖QEV‖QEU‖OFC_V MacTag1 = MAC(MacData1)$_{NK}$
MEA-3	U→V	MacTag2, OFC_U	
MEA-4	V→U	MacTag1, OFC_V	
Verification		MacData2 = 0x03‖U‖V‖QEU‖QEV‖OFC_U Verify MacTag2 using MacData2	MacData1 = 0x02‖V‖U‖QEV‖QEU‖OFC_V Verify MacTag1 using MacData1

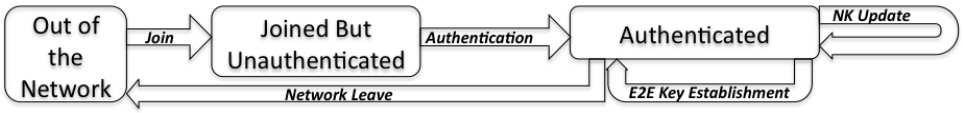


Figure 5: Security Procedures

previous knowledge to verify that they received the correct values, and thereby authenticate each other.

The MEA protocol uses two different frame formats for the challenge and the data, as shown in Fig. 3. The *MAC*, *Data Type*, and *Data* fields have the values field is the *MacTag*, *Frame Counter*, and the field is the *Frame Counter Value*, respectively.

4.9 CCM* Mode of Operation

CCM* is a generic encryption and authentication block cipher mode which is defined for use with only block ciphers having 128-bit block size. The AES-CCM* mode of operation is an extension of the AES-CCM mode that is used in IEEE 802.14.5-2003 [1] and provides capabilities for authentication, encryption, or both. Securing a NWK or an APS frame is actually based on AES-CCM* mode of operation in a particular security level.

5 The Security Procedures

In this section, we present a procedural description of how the security services in Section 4 are used. The security procedures consist of *joining a secured network*, *authentication*, *NK update*, *end-to-end application key establishment*, and *network leave*. We tried to visualize the security procedures in a state machine format as in Fig. 5. The arrows in the figure are the procedures, and the boxes are the states of a ZigBee device. Note that the procedure *End-to-End Key Establishment* is only valid in HS mode. Also note that it is also possible to change states without using procedures, as in the case of time outs, missed key update, etc.

5.1 Joining A Secured Network

This procedure is applied when a joiner device communicates with a router to join a secure network or when a device in a network has missed a key update and wants to receive the latest NK. The joiner device may begin the procedure by transmitting an unsecured beacon request frame. The joiner device receives beacons from routers and decides which network to join. After that the joiner device sends an association/rejoin request to the router. The router, knowing the joiner device's address and security capabilities (and in the case of *rejoin* whether the NK was used to secure the rejoin request command), will send an association/rejoin response command to the joiner device. If the joiner device receives a positive association/rejoin response command, the joiner is declared as *joined but unauthenticated* and the next phase shall be the Authentication routine. The Join procedure depends on the MAC layer command frames that are defined in the IEEE 802.15.4 standard, not in ZigBee Specification. Therefore, we did not include this procedure in the Appendix.

5.2 Authentication

A ZigBee device that successfully finished the *joining a secured network* procedure is declared as *joined but unauthenticated* and shall start the *authentication* procedure. If the device is not a router, then after successful completion of the authentication it will be declared as *joined and authenticated*. If the device is a router, then after successful completion of the authentication followed by the initiation of routing operations it will be declared as *joined and authenticated*.

A new feature of the ZigBee-2007 is that, not only the newly joined device but also the neighbouring routers must be authenticated by using the MEA protocol.

Authentication depends on many different parameters such as the security mode (SS, HS), the presence of a router between TC and the device, the security level (None, ENC, MIC, ENC-MIC), the preconfiguration of the device (Not preconfigured, Preconfigured with NK/MK/LK), profiles, etc.

In SS mode, there are three main scenarios:

NPK: Device is not preconfigured with a valid key: TC sends active NK to the device in plaintext.

PK-NK: Device is preconfigured with active NK: TC sends fake NK (all zeros) to the device in plaintext.

PK-TCLK: Device is preconfigured with TCLK: TC sends NK to the device, encrypted by TCLK (LK of the TC).

In HS mode, there are two main scenarios:

NPK: Device is not preconfigured with a valid key: According to its configuration, TC sends either TCLK or TCMK in plaintext (*We separated these cases as NPK-TCLK and NPK-TCMK, see Appendix*). If TC sends TCMK, then it also makes key establishment to derive a TCLK.

PK-TCMK: Device is preconfigured with TCMK: TC establishes TCLK with the device, using key establishment service.

In case of HS, the joiner establishes entity authentication with the router to complete the authentication procedure. In the case of a separate router existing between the TC and the device, the communication between the router and the TC is secured in APS layer using active NK in SS, TCLK in HS. The protocol narrations for the *Authentication* procedure is given in Appendix Subsection A.3.

5.3 NK Update

We have mentioned that, maintaining a list of all the devices in the network is optional in SS but mandatory in HS for a TC. In SS mode, TC broadcasts new SNK to all the devices. In HS mode, TC sends new HSNK to each device using unicast communication.

If the device is capable of storing an alternate NK (*i.e. FFD, see Section 2*), then it will replace its alternate NK with the new key it received. If the device is not capable of storing an alternate NK (*i.e. RFD, see Section 2*), then it will replace its current NK and ignore any Switch-Key command. In any case, all incoming frame counters and the outgoing frame counter of the appropriate NK shall be set to 0. The sequence number of the new key will be the sequence number of the previous key incremented by 1 in mod 256. The protocol narration for *NK Update* procedure is given in Appendix Subsection A.2.

5.4 End-to-End Application Key Establishment

When End-to-End Application Security between two devices is required, the devices will run this procedure which also requires the collabration of the TC. Depending on the configuration of the TC, the devices can either receive an application LK (**appLK**) or an application MK (**appMK**) from TC. In the case of receiving an appMK, it is necessary to establish an appLK afterwards, using the SKKE protocol. The protocol narration for the *End-to-End Application Key Establishment* procedure is given in Appendix Subsection A.4.

5.5 Network Leave

The Network Leave procedure actually works in two different ways: Remove-Device and Device-Leave. In *Remove-Device*, the TC wants a ZigBee device to be removed from the network. After TC informs the router about the situation, the router sends a leave command to the relevant device. In *Device-Leave*, a device decides to leave the network and sends a leave command to the router. Then, the router informs the TC about the situation. In either case, TC shall delete the device from its list (which is optional in SS mode). If TC and the router share a LK, then the messages between the two will be secured with LK, otherwise with NK. The messages between router and the leaving device will be secured by NK. The protocol narration for the *Network Leave* procedure is given in Appendix Subsection A.5.

6 Related Work

6.1 Public Key Cryptography and Key Establishment

The ZigBee Specification uses symmetric cryptography, and therefore lacks (*1*) session key distribution without the intervention of a key distribution center (trust center), (*2*) digital signature and the non-repudiation property. Public-Key (**PK**) cryptography could alleviate these shortcomings but one needs to consider the trade-off between the performance and the security. The *computational complexity* of symmetric cryptography is much less, whereas *key management* (the process of selecting, distributing and storing keys) is complex and relatively insecure compared to the PK-cryptography.

[11] is one of the oldest papers with a proposal for using PK-cryptography in ZigBee. It proposes dynamic establishment of the security keys between communicating nodes by using Elliptic Curve Cryptography (**ECC**) [10]. Although there are many PK systems such as RSA, DSA, and Diffie-Hellman, ECC is preferred because it requires shorter keys for the same level of protection.

[22] presents a computation time comparison of Elliptic Curve Diffie-Hellman (ECDH), Elliptic Curve Menezes-Qu-Vanstone (ECMQV) and Symmetric Key Agreement (SKA) for IEEE 802.15.4 networks. In addition, considering frame transmission time and encryption processing they claim that ECMQV is not much slower than SKA. In other words, providing better key management and security with a little more device overhead. Note that the SKA [23] is actually the same as SKKE, and ECMQV is included in the brand new ZigBee profile: Smart Energy (*see below*).

[13] proposes scalable key establishment protocols and secure routing protocols with scalable authentication schemes. The idea here is to use different protocols in different settings, i.e. pure symmetric cryptography for low security, hybrid (symmetric and PK) for medium, and pure PK-cryptography for high security. This is achieved by installing a group key and an ECC based public/private key pair into each device before they join the network, thus enabling both symmetric and PK-cryptography.

An application of the *identity-based cryptography* [20] which reduces the number of the required security keys in ZigBee networks is proposed in [19]. Identity-based cryptography has advantages over PK-cryptography because it eliminates the (need for the) public key directory. The key point here is using the identity of each entity as a public key, which results in a simplified key distribution process. However, this scheme has some disadvantages, i.e. when the private key of an entity is somehow compromised, changing the key pair is very problematic since the public key is the identity of the entity. [19] proposes a scenario in which the identity to be used is a concatenation of the device information such as function, location, timestamp etc. It also requires a private key generator which is actually the TC.

Reducing the high costs of the PK-cryptography in especially the RFD (*see Section 2*) end nodes is another important issue. [24] proposes a hybrid authenticated key establishment scheme where an implementation of ECC [25] and symmetric cryptography are combined. This

Table 8: CBKE Protocol (MAC: HMAC Function, ||: Concatenation, 0x: Hexadecimal)

		U	**V**
CBKE-1	U→V	S_U, E_U	
CBKE-2	V→U	S_V, E_V	
Computation		Z = KeyBitGen(S_U, E_U, S_V, E_V) KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) Mac_U=MAC(0x02,S_U,S_V,E_U,E_V)$_{MacKey}$	Z = KeyBitGen(S_U, E_U, S_V, E_V) KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) Mac_V=MAC(0x03,S_V,S_U,E_V,E_U)$_{MacKey}$
CBKE-3	U→V	MAC_U	
CBKE-4	V→U	MAC_V	
Verification		Verify Mac_V	Verify Mac_U
Use KeyData as the new LK			

combination reduces the required resources for computation in the RFD node by using symmetric key based operations. Besides, the usage of ECC avoids typical key management problems in pure symmetric key based protocols. [24] also shows that it is possible to have even more reduction by using Modular Square Root (MSR) [26] techniques instead of ECC.

ZigBee Smart Energy Profile Specification In the end of May 2008, ZigBee published the ZigBee Smart Energy Profile Specification [8], which seems to be their response to the PK-cryptography proposals. The Key Establishment Cluster defined in [8] states that two basic types of key establishment can be implemented:

1) *Symmetric Key Key Establishment (SKKE)*: Establishing a key (LK) based on a shared secret (MK). If the shared secret (MK) is compromised then established key (LK) also may be compromised.

2) *Public Key Key Establishment (PKKE)*: Establishing a key (LK) based on shared static and ephemeral public keys. The case that uses certificates signed by a Certificate Authority to transport static public keys is called Certificate-Based Key Establishment (CBKE). The Key Establishment Cluster uses CBKE with ECMQV key agreement and ECQV certificate generation [9].

The CBKE protocol is given in Table 8. S represents the static data, which is a combination of device address and device static public key, whereas E represents the ephemeral data. In the first two messages, the devices exchange static and ephemeral data. In the last two messages, the devices exchange the data they have computed using the static and ephemeral data. Note that ***KeyBitGen*** is the function for generating the ECMQV key bit stream, and *kdf* is the same key derivation function as in SKKE. After verifying that they received the correct values, they use another value as the LK that both of them can compute.

6.2 Vulnerabilities and Advices

The IEEE 802.14.5 standard and its potential security vulnerabilities are important since it is the structure that ZigBee is built on.

[21] pinpoints the problems in IEEE 802.14.5 security and classifies them as: (*1*) Initialization Vector (**IV**) Management Problems, (*2*) Key Management Problems, and (*3*) Integrity Protection Problems. Since the vulnerabilities may be avoided in different levels, the paper [21] also classifies the advice for these levels. The advice for application designers include avoiding usage of any security suite that does not have integrity protection (e.g. ENC), and for implementing their own acknowledgement system. The advice for hardware designers include improving Access Control List (ACL) and nonce usage, and eliminating the implementations of the security suites without integrity. Finally, the advice for specification/standard writers include necessary support, requirements and explanations for the vulnerable points which are pinpointed as problems.

[29] presents an attack classification based on layers. Jamming, capturing, tampering, exhaustion, collision, and unfairness are the attacks that are possible in the PHY and MAC layers. Routing disruption and resource consumption are the attack types that are possible in the NWK layer. In [29] some of these attacks are modelled in the network simulation system NS-2 [30] and the results are presented. In addition, relying heavily on the TC is an important criticism in the paper. Distributed or hierarchical key management schemes are recommended especially for large scale networks.

We believe that *key update* is an important issue in ZigBee networks. When a device is removed or leaving the network, it still knows the security key (i.e. NK). However, updating the NK after each device leave will be costly, whereas not updating the key will be insecure. Therefore, there is a trade-off between security and performance.

6.3 Surveys and Comparisons

[28] compares the security of two important Personal Area Network (PAN) standards, ZigBee and Bluetooth. It further claims that the secure key distribution and security key storage are the issues that may have problems in the future.

Z-Wave is a rival technology to ZigBee, which is a proprietary one developed by a Danish company. [12] presents a comparison of ZigBee and Z-Wave, and discusses security issues.

[14] compares ZigBee security with other security architectures in Wireless Sensor Networks (**WSN**) such as SPINS [16], LEAP [18], TINYSEC [17] and SM [15]. In this work, authentication characteristics are focused and it is concluded that the trend has moved from pre-deployed keying mechanisms to the symmetric key agreements, then to ECC based algorithms to perform authentication in WSNs.

[37] reviews five key management schemes: Eschenauer [38], Du [39], LEAP, SHELL [40] and PANJA [41]. This study evaluates this schemes considering the new trends in IEEE 802.15.4 and ZigBee. They conclude that future developments could incorporate the flexibility of LEAP with the adjustable robustness of Du or Eschenauer. In addition, for highest robustness Shell, and for highly scalability Panja can be preferred.

Claiming that being the first study that analyses SKKE, [27] presents a performance analysis of the key exchange mechanism. Their results indicate that even for small network size frequent key exchanges impose a serious performance burden on the data traffic.

As we mentioned before, TINYSEC is a WSN protocol that can be compared to ZigBee. [35] claims that, TINYSEC achieves low energy consumption by low security, ZigBee suffers from high energy consumption by high security, whereas their new proposal MiniSec obtains low energy consumption and high security. The idea behind this assertion is employing Offset Codebook Mode (OCB) [36] in encryption, sending only a few bits of the IV, and having different energy-optimized communication modes for unicast and broadcast communication.

Although the issue of contention resolution is beyond this survey, [34] is an interesting work since it presents the first application of *probabilistic model checking* to the IEEE 802.15.4 standard.

[33] is a recent extensive survey on WSNs; however the ZigBee section covers ZigBee-2006 only, like all other references. In addition, [42] is also a useful survey for WSNs, although it is not directly related to ZigBee.

6.4 Burglary Protection

Burglary protection is an interesting topic in pervasive computing. The attacks on security protocols are not the only threats in ZigBee networks, *theft* is also a serious security threat.

[31] proposes a security policy which is an extension of a more general model called "Resurrecting Duckling" [32]. The main idea in [31] is to chain the devices in a network or in friendly networks in such a way that a device will only function when it can see all of its friends. This idea is realized with protocols defined for device association and presence verification, and some real life scenarios are presented in the paper.

7 Conclusion

ZigBee is an emerging wireless network standard with low resource requirements. The latest version of the ZigBee Specification, ZigBee-2007, enhances the security of ZigBee. In this paper we presented a high level self-contained overview of the ZigBee-2007 security. We explained the key points of the specification such as *security in different layers*, presented the computations behind key establishment and authentication schemes such as *SKKE*, *CBKE*, *MEA*, and went deeper into the essential protocol narrations such as *Authentication*, *NK Update*, *etc.* We expect that ZigBee's wide applicability in many areas will necessitate much more work in ZigBee security. We plan to extend our work with the analysis and verification of the ZigBee security protocols.

References

[1] *IEEE Std. 802.15.4-2003, Wireless Medium Access Control and Physical Layer Specifications for Low Rate Wireless Personal Area Networks*, IEEE, 2003.

[2] *ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American Bankers Association, 2001.

[3] *FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197*, US Dpt. of Commerce/N.I.S.T, 2001.

[4] *FIPS Pub 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198*, US Dpt of Commerce/N.I.S.T., 2002.

[5] *ZigBee Specification*, ZigBee Alliance, r17, January 2008.

[6] *ZigBee Specification*, ZigBee Alliance, r13, October 2006.

[7] *ZigBee Specification*, ZigBee Alliance, r06, December 2004.

[8] *ZigBee Smart Energy Profile Specification*, ZigBee Alliance, r14, May 2008.

[9] *Standards for Efficient Cryptography: SEC 1 (working draft) ver 1.7: Elliptic Curve Cryptography*, Certicom Research, November 2006.

[10] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, 48, pp. 203-209, 1987.

[11] M. Blaser, "Industrial-strength Security for Zigbee: The Case for Public-key Cryptography", *Embedded Computing*, 2005.

[12] M. Knight, "How safe is Z-wave?", *Engineering and Technology*, 2006.

[13] Q. Huang, H. Kobayashi, B. Liu, D. Gu, and J. Zhang, "Energy/Security Scalable Mobile Cryptosystem", *Proc. IEEE Int'l Symp. on Personal, Indoor and Mobile Radio Comm.*, Vol. 3, pp. 2755-2759, September 2003.

[14] D. Boyle, and T, Newe, "Security Protocols for Use with Wireless Sensor Networks: A survey of security architectures", *Proc. Int'l Conf. on Wireless and Mobile Comm.*, 2007.

[15] J. Heo, C.S. Hong, "Efficient and Authenticated Key Agreement Mechanism in Low-Rate WPAN Environment", *Proc. IEEE Int'l Symp. on Wireless Pervasive Computing*, 2006.

[16] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and D. Culler, "SPINS: Security Protocols for Sensor Networks", *Wireless Networks*, 8(5), pp. 521-534, 2002.

[17] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks", *Proc. 2nd ACM Int'l Conf. on Embedded Networked Sensor Systems*, pp. 162-175, 2004.

[18] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks", *Proc. ACM Conf. on Computer and Comm. Security*, pp. 62-72, October 2003.

[19] S.T. Nguyen, and C. Rong, "ZigBee Security Using Identity-Based Cryptography", *Lecture Notes in Computer Science*, 4610, pp. 3-12, 2007.

[20] A. Shamir, "Identity-based Cryptosystems and Signature Schemes", In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47-53. Springer, Heidelberg, 1985.

[21] N. Sastry, and D. Wagner, "Security Considerations for IEEE 802.15.4 Networks", *Proc. 3rd ACM Workshop on Wireless security*, October 2004.

[22] J. Heo, and C.S. Hong, "Efficient and Authenticated Key Agreement Mechanism in Low-Rate WPAN Environment", *Proc. 1st IEEE Int'l Symp. Wireless Pervasive Computing*, pp. 30-34, January 2006.

[23] *ISO/IEC 9798-2, Information Technology - Security Techniques - Entity Authentication Mechanism - Part 2, "Mechanisms Using Symmetric Encipherment Algorithm"*, International Standardization Organization, 1994.

[24] Q. Huang, J.I. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast Authenticated Key Establishment Protocols for Self-Organizing Sensor Networks", *Proc. Int'l Conf. on Wireless Sensor Networks and Applications*, pp. 141-150, September 2003.

[25] *Standard for efficient cryptography, SEC 1: Elliptic Curve Cryptography. Version 1.0*, September 20, 2000. Certicom Corporation.

[26] D.R. Stinson, "The Rabin Cryptosystem, Cryptography: Theory and Practice", Section 4.7, CRC Press (1995).

[27] M. Khan, F. Amini, J. Misic, and V. B. Misic, "The Cost of Security: Performance of ZigBee Key Exchange Mechanism in an 802.15.4 Beacon Enabled Cluster", *Proc. 3rd IEEE Int'l Conf. on Mobile Ad Hoc and Sensor Systems*, pp. 876-881, October 2006.

[28] O. Hyncica, P. Kacz, P. Fiedler, Z. Bradac, P. Kucera, and R. Vrba, "On Security of PAN Wireless Systems", *Lecture Notes in Computer Science*, pp. 178-185, 2006.

[29] J. Zheng, M.J. Lee, and M. Anshel, "Toward Secure Low Rate Wireless Personal Area Networks", *IEEE Trans. on Mobile Computing*, Vol. 5, No. 10, pp. 1361-1373, 2006.

[30] Network Simulator - NS2, USC Information Sciences Institute, 2005.

[31] M.Ø. Pedersen, J.I. Pagter, "The All-Or-Nothing Anti-Theft Policy: Theft Protection for Pervasive Computing", *Proc. IEEE Computer Society AINA Workshops*, pp. 626-631, 2007.

[32] F. Stajano and R. Anderson. "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks", *Proc. 7th Int'l Workshop Security Protocols*, 1999.

[33] P. Baronti, P. Pillai, V.W.C. Chook, S. Chessa, A. Gotta, and Y.F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards", *Computer Communications*, v.30 n.7, pp.1655-1695, May 2007.

[34] M. Fruth, "Probabilistic Model Checking of Contention Resolution in the IEEE 802.15.4 Low-Rate Wireless Personal Area Network Protocol", *Proc. 2nd Int'l Symp. on Leveraging Applications of Formal Methods, Verification and Validation*, pp. 290-297, November 2006.

[35] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Comm. Architecture", *Proc. IEEE Int'l Conf. on Inf. Processing in Sensor Networks*, April 2007.

[36] P. Rogaway, M. Bellare, and J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption", *ACM Trans. on Information and System Security*, vol. 6, no. 3, pp. 365-403, August 2003.

[37] J.C. Lee, K.H. Wong, J. Cao, H.C.B. Chan and V.C.M. Leung, "Key management issues in wireless sensor networks: current proposals and future developments", *IEEE Wireless Comm.*, April 2007.

[38] L. Eschenauer and V.D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks", *Proc. 9th ACM Conf. Comp. and Comm. Sec.*, pp. 41-47, 2002.

[39] W. Du, J. Deng, Y.S. Han, P.K. Varshney, J. Katz, and A. Khalili, "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks", *Proc. 10th ACM Conf. Comp. Comm. Sec.*, pp. 42-51, 2003.

[40] M. F. Younis, K. Ghumman, and M. Eltoweissy, "Location-Aware Combinatorial Key Management Scheme for Clustered Sensor Networks", *IEEE Trans. Parallel and Distrib. Sys.*, vol. 17, pp. 865-82, 2006.

[41] B. Panja, S.K. Madria, and B. Bhargava, "Energy and Communication Efficient Group Key Management Protocol for Hierarchical Sensor Networks", *Proc. IEEE Conf. Sensor Networks, Ubiquitous, and Trustworthy Comp.*, pp. 384-93, 2006.

[42] J.P. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless Sensor Network Security: A Survey", *Security in Distributed, Grid, Mobile, and Pervasive Computing*, Ed. Y. Xiao. Auerbach Publications, CRC Press, 2007.

[43] J. Zheng, and M.J.Lee, "Will IEEE 802.15.4 Make Ubiquitous Networking a Reality?- A Discussion on a Potential Low Power, Low Bit Rate Standard", *IEEE Comm. Magazine*, June 2004.

A Protocol Narrations

A.1 Protocol Narration Convention

Extending the classical *Alice-Bob* protocol narration style, we used message (primitive) titles in boldface fonts, and message fields inside angle brackets. The protocol narration convention is given below:

Format: A. B→C: D-E.F
<M>

A: Message Number (*1,2,3,...*)
B and C: Sender and Receiver (Trust Center: *TC*, Router: *R*, All Routers: *Rn*, Device: *D*, All Devices: *Dn*, Initiator: *I*, Responder: *Res*)
D: Message Layer and Entity (APS Management Entity: *APSME*, NWK Management Entity: *NLME*)
E: Command Type (*SWITCH-KEY, ESTABLISH-KEY, etc.*)
F: Primitive Type (*request, response*)
M: Message structure, composed of fields (*Accept, SrcAddress, etc.*).

Example:
3. D→TC: APSME-ESTABLISH-KEY.response
<InitiatorAddress, Accept>

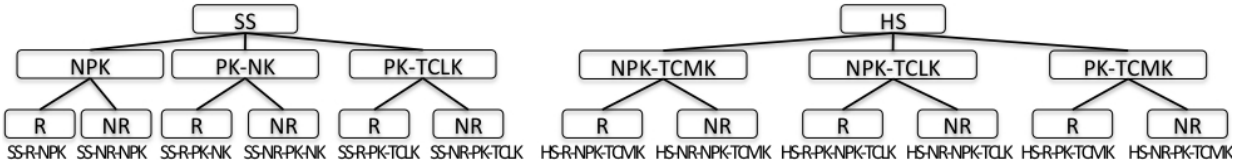


Figure 6: Authentication Protocols

This is the third message of the protocol, sent by a device to the TC, the message is an APS layer management entity message, the command type is *establish key*, the primitive type is *response*. The message has two fields, *InitiatorAddress* and *Accept*.

A.2 NK Update

We have two different NK Update protocols for SS and HS modes.

SS	**HS**
1. TC→Dn: APSME-TRANSPORT-KEY.request <DestAddress, KeyType, KeySeqNumber, NetworkKey, UseParent, ParentAddress> **2. TC→Dn: APSME-SWITCH-KEY.request** <DestAddress, KeySeqNumber>	**1. TC→D: APSME-TRANSPORT-KEY.request** <DestAddress, KeyType, KeySeqNumber, NetworkKey, UseParent, ParentAddress> **2. TC→D: APSME-SWITCH-KEY.request** <DestAddress, KeySeqNumber>

A.3 Authentication

Protocol Naming Convention In order to have a shorter and a systematic way of naming protocols, we created the protocol naming convention below for the Authentication protocols:

Format: G-H-I[-J]
G: Security Mode (Standard Security: *SS*, High Security: *HS*)
H: Presence of a separate router (A separate router exists between TC and D: *R*, TC serves as the router: *NR*)
I: Key knowledge of the device (Preconfigured with a key: *PK*, Not preconfigured: *NPK*)
J: Preconfigured or "to be configured" key type
(None, Network Key: *NK*, Trust Center Link Key: *TCLK*, Trust Center Master Key: *TCMK*)

Example: SS-R-PK-TCLK means: Standard Security mode, TC is not the router, Device is preconfigured with TCLK

The protocols in the Authentication part can be summarized as in Fig. 6. In order to save space, we present the narrations in the case that TC is the router.

SS-NR-NPK
1. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress>

SS-NR-PK-NK
1. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress>

SS-NR-PK-TCLK
1. TC→D: APSME-TRANSPORT-KEY.request
<{DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress}:TCLK>

HS-NR-NPK-TCMK
1. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, ParentAddress, Key>
2. TC→D: APSME-ESTABLISH-KEY.request
<ResponderAddress, UseParent, ResponderParentAddress, KeyEstablishmentMethod>
3. D→TC: APSME-ESTABLISH-KEY.response
<InitiatorAddress, Accept>
4. TC↔D: SKKE (*See Subsection 4.7 for details*)
5. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress>
6. D→TC: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
7. TC→D: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
8. D↔TC: MEA (*See Subsection 4.8 for details*)

HS-NR-NPK-TCLK
1. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, ParentAddress, Key>
2. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress>
3. D→TC: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
4. TC→D: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
5. D↔TC: MEA (*See Subsection 4.8 for details*)

HS-NR-PK-TCMK
1. TC→D: APSME-ESTABLISH-KEY.request
<ResponderAddress, UseParent, ResponderParentAddress, KeyEstablishmentMethod>
2. D→TC: APSME-ESTABLISH-KEY.response
<InitiatorAddress, Accept>
3. TC↔D: SKKE (*See Subsection 4.7 for details*)
4. TC→D: APSME-TRANSPORT-KEY.request
<DestAddress, Keytype, KeySeqNumber, NetworkKey, UseParent, ParentAddress>
5. D→TC: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
6. TC→D: APSME-AUTHENTICATION.request
<PartnerAddress, Action, RandomChallenge>
7. D↔TC: MEA (*See Subsection 4.8 for details*)

A.4 End-to-End Application Key Establishment

This procedure depends on the configuration of the TC. Therefore, we have two different protocols.

TC is configured to send AppLK
1. I→TC: APSME-REQUEST-KEY.request
<DestAddress, KeyType, PartnerAddress>
2. TC→I: APSME-TRANSPORT-KEY.request
<DestAddress, KeyType, PartnerAddress, Initiator, Key>
3. TC→Res: APSME-TRANSPORT-KEY.request
<DestAddress, KeyType, PartnerAddress, Initiator, Key>

TC is configured to send AppMK
1. I→TC: APSME-REQUEST-KEY.request
<DestAddress, KeyType, PartnerAddress>
2. TC→I: APSME-TRANSPORT-KEY.request
<DestAddress, KeyType, PartnerAddress, Initiator, Key>
3. TC→Res: APSME-TRANSPORT-KEY.request
<DestAddress, KeyType, PartnerAddress, Initiator, Key>
4. I→Res: APSME-ESTABLISH-KEY.request
<ResponderAddress, UseParent, ResponderParentAddress, KeyEstablishmentMethod>
5. Res→I: APSME-ESTABLISH-KEY.response
<InitiatorAddress, Accept>
6. I↔Res: SKKE

A.5 Network Leave

This procedure has different protocols, depending on the initiator of the procedure.

Remove-Device
1. TC→R: APSME-REMOVE-DEVICE.request
<ParentAddress, ChildAddress>
2. R→D: NLME-LEAVE.request
<DeviceAddress, RemoveChildren, Rejoin>

Device-Leave
1. D→R: NLME-LEAVE.request
<DeviceAddress, RemoveChildren, Rejoin>
2. R→TC: APSME-UPDATE-DEVICE.request
<SrcAddress, DeviceAddress, Status, DeviceShortAddress>

Protocol Analysis in a new LyTE

Nicholas O'Shea
School of Informatics,
Laboratory for Foundations of Computer Science,
The University of Edinburgh, Scotland.

Abstract

Cryptographic protocol analysis is a heavily studied field yet underused by the majority of software developers. In this paper we discuss our plugin for the Eclipse IDE for the LySa process calculus which provides several tools to help developers understand, edit and develop new protocol models. We also show how tools included in this plugin can be used to validate the output of the Elyjah tool[12] which converts Java implementations into LySa models.

1 Introduction and Motivation

Cryptographic protocols continue to be a widely studied field in computer security due to the ever increasing demand for secure communication and reliable authentication. Before becoming an accepted standard, protocol models are analysed, debated and re-proposed by security experts the world over. Even after this process and years of use, it is not unheard of for a new flaw to be found in the protocol which undermines its functionality. The trouble is that while the protocols themselves are usually concise, needing only a few messages to terminate, the flaws can be very subtle. Automated analysis is the most efficient method for analysing cryptographic protocol models. There are numerous techniques and tools for achieving this purpose with more devised every year[1, 9].

In the past, few applications needed to utilise networking properties. Thus the development of these applications could be overseen by a security expert, versed in the potential flaws of cryptographic protocols. These days however, it is easier than ever to develop an application which requires secure network functionality. There is also more reason to do so. Businesses are more aware of cyber-espionage and the average individual has more understanding of the lack of privacy. With mobile devices becoming more powerful with a greater ability for applications to run on them, the potential demand for lightweight but secure communication has never been higher. The average developer for these applications has access to integrated development environments, garbage collection and managed code turning software development into an exercise that requires less thought than ever. When the flaws in cryptographic protocols can be as subtle as they often are, this is not necessarily a good thing. More developers means more mistakes. As many of these developers will not have significant education on cryptographic protocols, mistakes of the past may be repeated. Most developers are unfamiliar with the vast array of methods for analysing protocols and, even if they

were made aware, are unlikely to be sufficiently motivated to learn how to use them given other, greater, pressures on their time such as release deadlines.

Even assuming a developer researches the topic and chooses to implement a protocol which has been verified by one or several security experts there are still a number of potential problems. They may be working from a faulty specification, they may misunderstand the specification, they may even make the kind of basic implementation error that anyone can suffer. For this reason, it is more important than ever to analyse *implementations* of protocols rather than to be content just to analyse a model and assume that a developer will implement it correctly.

In this paper we present work which is aimed at helping inexperienced developers to utilise the benefits of formal cryptographic protocol analysis. This is achieved by making it easier to understand and use the formal methods as well as providing techniques that enable analysis without the user understanding any of the formal methodology behind it. The goal of this work is to provide developers with tools that will assist in the development of secure networking applications. In particular, these tools should be in use during the development phase of an application reducing the need for continued updating and patching of software. The techniques used by these sorts of tools should help to find security breaches and allow both developers and users to feel confident that the integrity of the network will not be compromised. If we can verify code in the languages which implementers actually use, we can find and fix security properties as soon as protocols are implemented.

2 LySa and the LySatool

2.1 LySa

The process calculus LySa[4] is similar to the Spi-calculus in that it defines the actions of the principals involved in the protocol. A crucial difference is that LySa does not use the concept of channels to send messages. Instead LySa assumes there is a global medium through which all principals communicate. The designers of LySa took this approach as they believed that private channels provided a layer of privacy not matched in a real world scenario where attackers can eavesdrop or add in messages of their own. The syntax of LySa terms, E, can be found in Figure 1.

$E ::=$	$terms$
n	name($n \in \mathcal{N}$)
x	variable($x \in \mathcal{X}$)
$\{E_1,\ldots,E_k\}_{E_0}$	symmetric encryption
$\{\|E_1,\ldots,E_k\|\}_{E_0}$	asymmetric encryption

Figure 1: Syntax of LySa terms

Encrypted messages are tuples of terms encrypted under a key, E_0. LySa assumes a perfect encryption system that can only be broken by knowing the correct decryption key. Both symmetric and asymmetric encryption are supported. The syntax of processes, P, can be found in Figure 2.

$P ::=$	*processes*
0	termination process
$\langle E_1,\ldots,E_k\rangle.P$	output
$(E_1,\ldots,E_j;x_{j+1},\ldots,x_k).P$	input (with matching)
$P_1 \mid P_2$	parallel composition
$(\nu n)P$	name creation
$(\nu \pm m)P$	key pair creation
$!P$	replication
decrypt E as $\{E_1,\ldots,E_j;x_{j+1},\ldots,x_k\}_{E_0}$ in P	symmetric decryption (with matching)
decrypt E as $\{\|E_1,\ldots,E_j;x_{j+1},\ldots,x_k\|\}_{E_0}$ in P	asymmetric decryption (with matching)

Figure 2: Syntax of LySa processes

Another useful aspect of LySa is the pattern matching that is applied to incoming messages. Decryption has the same pattern matching ability and is performed in the same manner. The first j parts have to match with constants and the remaining $k - j$ parts are bound to variables. In order to use pattern matching to analyse a protocol, it may be necessary to reorder the contents of a message so that the parts that the receiver checks are all grouped together at the start of the message. For example in the second protocol given in [11], the third message reads

$$\text{A} \rightarrow \text{B } \{\text{I}_A, \text{A}\}^{PKB}$$

However, in the LySa formal model, the recipient of the message(the principal designated as B) first checks that the message is sent by A and stores the incoming nonce I_A in a local variable. However, to do this the message needs to be reordered so the nonce is the second part of the message. The LySa processes for both sending and receiving the message are as follows.

Sending	Receiving
$\langle$A, B, $\{\|$A, $\text{I}_A\|\}_{PKB+}\rangle$.	(A, B; y). decrypt y as $\{\|$A; nonce$\|\}_{PKB-}$ in 0

Crypto-points are a vital part of LySa's ability to be used to analyse protocols. Without them the LySatool would not be able to report any violations of the protocol's secrecy. A crypto-point represents a site in the protocol where either an encryption or decryption takes place. Coupled with a crypto-point is an assertion about the origin or destination of the encrypted message. When part of a message is encrypted or decrypted, the developer can choose to specify the current location as a crypto-point. Additionally, for an encryption, the developer can then choose to specify one or more crypto-points where decryption should take place during a valid run of the protocol. This is done like this:

$$[\text{at } a \text{ dest } \{b\}]$$

The crypto-point for the corresponding decryption specifies where the message should have been encrypted and is as follows:

[at b orig $\{a\}$]

The analysis performed by the LySatool is concerned with establishing the validity of these assertions. The following LySa model represents the Otway-Rees protocol.

(ν KAS)((ν KBS)(
(!(ν NA)
⟨A, B, M, A, B, $\{$M, A, B, NA$\}_{KAS}$ [at a1 dest $\{$s1$\}$]⟩.
(B, A, M; x1). decrypt x1 as $\{$NA; xk$\}_{KAS}$ [at a2 orig $\{$s3$\}$] in
(B, A; x2). decrypt x2 as $\{$; xmsg$\}_{xk}$ [at a3 orig $\{$b3$\}$] in 0)
|
(!(ν NB)
(A, B, M, A, B; y1).
⟨B, S, M, A, B, y1, $\{$M, A, B, NB$\}_{KBS}$ [at b1 dest $\{$s2$\}$]⟩.
(S, B, M; y2, y3). decrypt y3 as $\{$NB; yk$\}_{KBS}$ [at b2 orig $\{$s4$\}$] in
⟨B, A, M, y2⟩.(new MSG) ⟨B,A, $\{$MSG$\}_{yk}$ [at b3 dest $\{$a3$\}$]⟩.0)
|
(!(B, S, M, A, B; z1, z2).
decrypt z1 as $\{$M, A, B; zna$\}_{KAS}$ [at s1 orig $\{$a1$\}$] in
decrypt z2 as $\{$M, A, B; znb$\}_{KBS}$ [at s2 orig $\{$b1$\}$] in (new K)
⟨S, B, M, $\{$zna, K$\}_{KAS}$ [at s3 dest $\{$a2$\}$], $\{$znb, K$\}_{KBS}$ [at s4 dest $\{$b2$\}$]⟩.0)))

2.2 LySatool

The LySatool is an automatic tool for checking the security properties of protocols. The tool inputs protocols modelled in the LySa process calculus. It then provides feedback regarding which message parts can be decrypted as well as whether an attacker can falsely achieve authentication by inserting messages at any point. An important point to mention is that the analysis is performed on an abstraction of the actual behaviour. After analysis it must be possible to map the result back to the semantics, a process called concretisation. As the analysis only represents certain aspects of a process' behaviour this concretisation leads to imprecision with respect to the originally analysed process. This over approximation can guarantee confidentiality with the downside that the tool can potentially report faults due to attacks that are impossible to reproduce in a real world scenario. Such faults have not yet surfaced in our practical examples.

3 Elyjah

In [12] we introduced a tool which translates Java implementations of cryptographic protocols into a LySa model. This work will be summarised in the following section. Elyjah was developed to be a compilation time tool to allow developers attempting to implement a protocol to receive assurances that they had both correctly implemented the specification and that the protocol was secure. In order to be able to verify implementations the first stage was to design and implement a new API for message composition and communication. Although it would be possible for a software tool

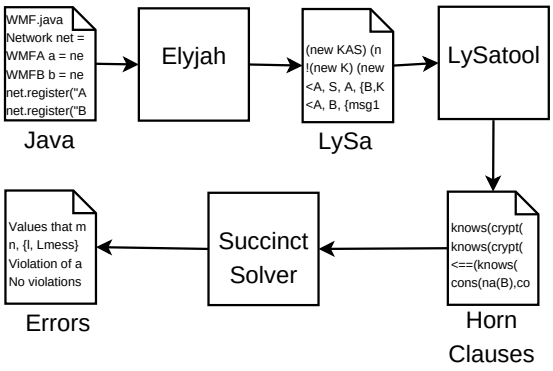


Figure 3: Analysis performed on a Java implementation of a protocol

to be able to accept any Java program as input before converting the source code into a LySa process, it would be substantially less reliable than one which demands a uniform input format. Thus a framework is needed to allow developers to model protocols, while also allowing the software tool to parse the source code and identify the security-critical operations. An implementation of a protocol needs to be a fully-working Java program which the developer can use to test that the protocol functions as expected before analysing its security properties. In order for it to be possible for any tool to understand the developer's intent, there needs to be a standard method of achieving certain goals. As such there needs to be set ways of encrypting/decrypting data as well as sending messages between principals. This API gives a developer easy access to cryptographic and communication functions and makes it easy for an automated tool to find and analyse these functions.

Elyjah identifies method calls to the provided API. Using these method calls as signposts, the cryptographic and communication parts of the security protocol can be divined. This is required as Java is more expressive than LySa so the language needs to be restricted in order to be able to achieve accurate translation. This means that Elyjah allows a Java implementation of a cryptographic protocol to be translated into a formal model. The LySa model can be analysed using the LySatool to return the security properties of the protocol implementation. While Elyjah cannot analyse the source code of an arbitrary communications package, it is possible to implement a protocol that can be translated by Elyjah and used in a full application. Thus, in order to analyse a Java implementation of a protocol there is a three step procedure as described in Figure 3. As both the conversion from Java to LySa and the analysis of the LySa model are performed by static analysis the whole process runs very quickly. Thus is suitable for use as a compilation-time tool that will not slow the progress of the developer(meaning it is more likely they will actually use it). Elyjah is provided as a plugin for the Eclipse IDE[7] which is a popular development environment for the Java language. The LySa output from Elyjah and the analysis generated by the LySatool are displayed to the user in an Eclipse console(Figure 4), similar to standard compilation problems. The developer may also save the LySa model so that it can be edited and analysed by the tools described in Section 4. The Java implementations are fully-functioning programs that can be run locally or distributed. When run they provide

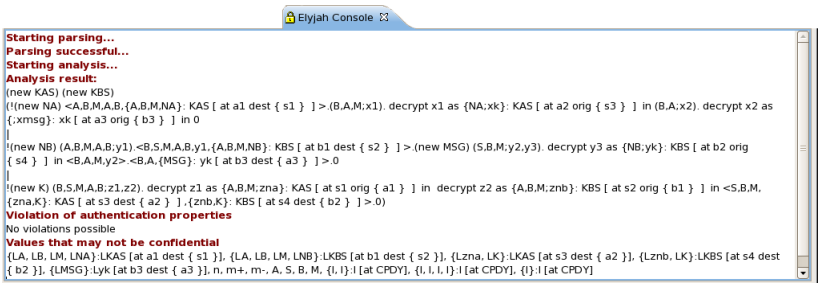


Figure 4: Example output of Elyjah

an execution trace of what is being sent in typical protocol narration form. Encrypted parts of the message merely provide the structure of the encrypted string for privacy reasons. The contents of the variables are also shown instead of any variable names.

4 LyTe

The LySa Toolkit in Eclipse(LyTE) was designed to help developers work with the LySa code generated by Elyjah. It aims to make LySa slightly less intimidating to any developers who were introduced to the process calculus through Elyjah. Like Elyjah it is also implemented as an Eclipse plugin so much of the operation will be familiar to developers who should not have to learn how to use a new tool in order to learn a new language. A screenshot of LyTE running in Eclipse is presented in Figure 5.

4.1 LySa Editor

The LySa Editor is designed to help developers write and edit LySa models. It features dynamic parse checking to identify errors speedily and provide the developers with a useful error message and suggested fixes. Fitting in to the Eclipse platform, the error is underlined, the line marked, and an error message added to the problem console. This checking can identify syntax flaws such as using the wrong punctuation, misspelt key words or missing pattern matching on receive and decrypt messages. The editor also provides syntax highlighting, which helps separate send, receive and decrypt processes by highlighting them in different colours. There is additional static analysis which checks that for each send process there is a corresponding possible receive process with the same source and destination as well as the right number of tuples. This provides a weak form of liveness checking that warns users of possible areas of deadlock in their models. The user's LySa model can then be analysed with the LySatool, the results of which are displayed in an Eclipse console.

4.2 AnaLySa

This tool translates a LySa model into standard protocol narration. Standard protocol narrations are often used to describe protocols, although the trouble is that they only contain half of the protocol. Namely they do not say what the recipient does upon receiving a message. However, they are used because they succinctly describe the

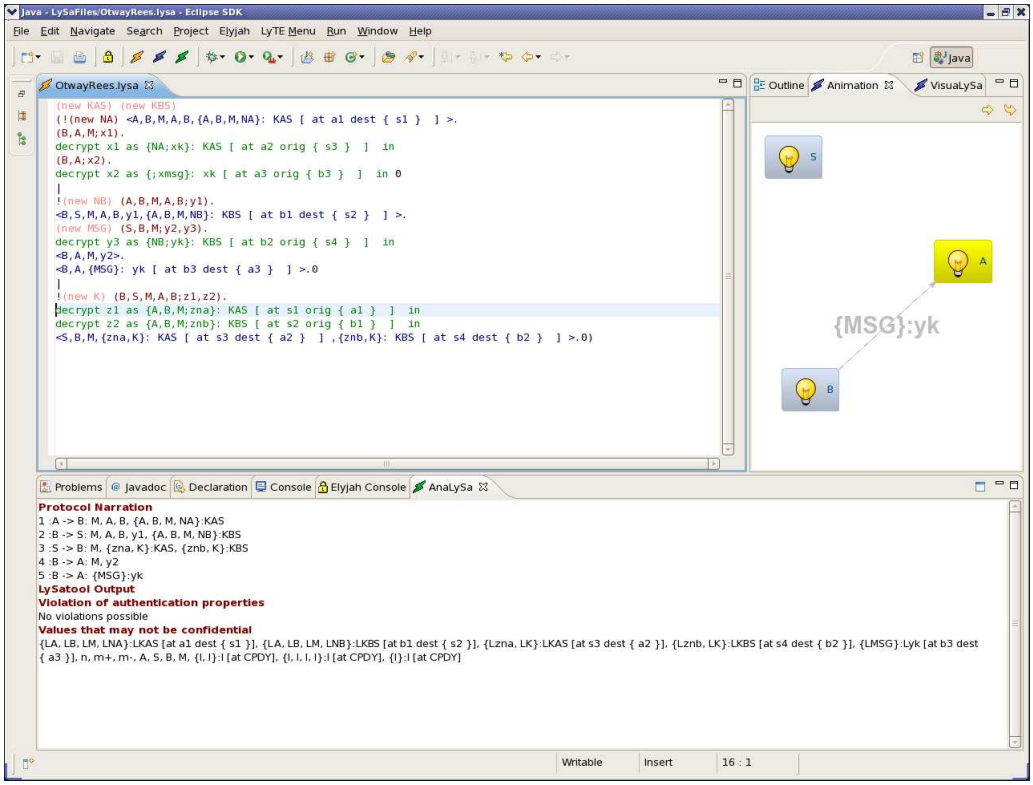


Figure 5: The LySa Toolkit in Eclipse

messages exchanged. Figure 6 shows a typical protocol narration, namely the Otway-Rees protocol.

$$\begin{aligned}
&A \rightarrow B : M, A, B, \{A, B, M, N_A\}_{K_{AS}} \\
&B \rightarrow S : M, A, B, \{A, B, M, N_A\}_{K_{AS}}, \{A, B, M, N_B\}_{K_{BS}} \\
&S \rightarrow B : M, \{N_A, K\}_{K_{AS}}, \{N_B, K\}_{K_{BS}} \\
&B \rightarrow A : M, \{N_A, K\}_{K_{AS}} \\
&B \rightarrow A : \{MSG\}_K
\end{aligned}$$

Figure 6: Otway-Rees protocol narration

A protocol narration is generated from a LySa process by creating a table of LySa send processes and the receive process that comes before each one. This enables the order of send processes to be worked out, starting by looking for the send process with no previous receive command. After the first message is identified, the matching receive process for this message is found and using the previously-generated table the next send process can be discovered. This process continues for all send processes. Below is the result from the AnaLySa on a LySa model of the Otway-Rees protocol.

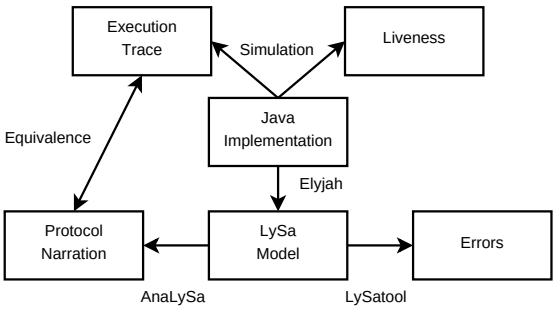


Figure 7: Using LyTe to validate Elyjah output

```
Protocol Narration
1 :A -> B: M, A, B, {A, B, M, NA}:KAS
2 :B -> S: M, A, B, y1, {A, B, M, NB}:KBS
3 :S -> B: M, {zna, K}:KAS, {znb, K}:KBS
4 :B -> A: M, y2
5 :B -> A: {MSG}:yk
```

4.2.1 Elyjah Translation Validation

The AnaLySa can be used to act as a simple form of Translation Validation[13] for the Elyjah tool. The output from the AnaLySa can be compared with the execution trace generated by running the Java implementation. Both versions of the protocol narration should have the same structure. This acts as a case-by-case check of Elyjah's translation. It is said that there are two ways to prove the correctness of a tool like Elyjah, either proving the tool or proving the output. Using the AnaLySa allows the developers to perform a simple proof of the correctness of the LySa model in relation to the Java implementation. Here we present the execution trace from the Java implementation that generated the LySa model of the Otway-Rees protocol used in Section 4.2.

```
18-Jul-2008 11:12:09 Network send
INFO: 1 :A -> B: 1, A, B, {M0, M1, M2, M3}:KAS
18-Jul-2008 11:12:09 Network send
INFO: 2 :B -> S: 1, A, B, 124AFog5PqCr/HCsdZ/5Jg==,
  {M0, M1, M2, M3}:KBS
18-Jul-2008 11:12:09 Network send
INFO: 3 :S -> B: 1, {M0, M1}:KAS, {M0, M1}:KBS
18-Jul-2008 11:12:09 Network send
INFO: 4 :B -> A: 1, qirLTAOPhn5FmkkY70GBtN8rIUSLgJCf
18-Jul-2008 11:12:09 Network send
INFO: 5 :B -> A: {M0}:yk
```

Comparing the two different protocol narrations reveal the same structure thus providing evidence of correct translation for this example. However, there are a few key differences. Firstly the serial number, M in the AnaLySa result, is replaced with the actual number in the execution trace. Use of encryption results in two further differences,

firstly the execution trace only contains the structure of the encrypted section and secondly the fourth tuple in the second message and the second tuple in the penultimate message are the encrypted string rather than the variable names used to identify them. This system has a couple of limitations. Firstly as we are comparing standard protocol narrations we are only checking that the messages that are sent are the same. There are no guarantees that the message is processed the same way in the LySa version as in the original implementation. Further work on this tool will correct this shortcoming. At the moment however, we believe it provides a useful reasurrance to developers that Elyjah is correctly translating their Java implementation.

4.3 VisuaLySa

An alternative to the protocol narration is a visual representation of the protocol. This option is provided by the VisuaLySa which translates a LySa process into a sequence-diagram style diagram. This sort of option is useful for developers who are playing around with LySa and trying to learn how it works through experimentation. An example of VisuaLysa is shown in the generated visual representation of the Otway-Rees protocol in Figure 8.

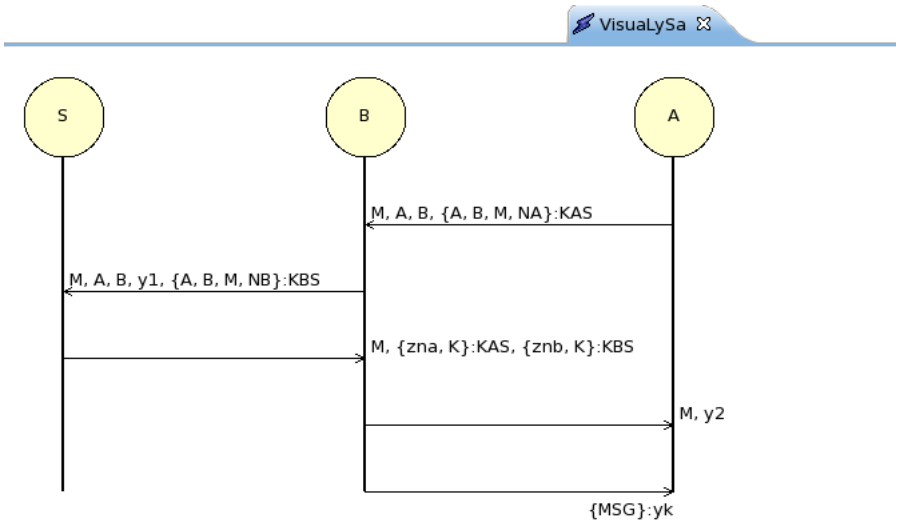


Figure 8: Example output of VisuaLysa

4.4 Animation

A different graphical representation of a protocol is provided as well as the sequence diagram. This is an animation of the messages sent between various principals which can be run as an animation or by stepping through in a single step manner. This is seen in Figure 9. After the protocol is terminated the diagram shows the flow of communication that occurred during the protocol run. Along with the AnaLySa described above, the VisuaLySa provides a useful function of particular interest to those new to LySa or other process calucli. Often a new developer has no way of checking that their

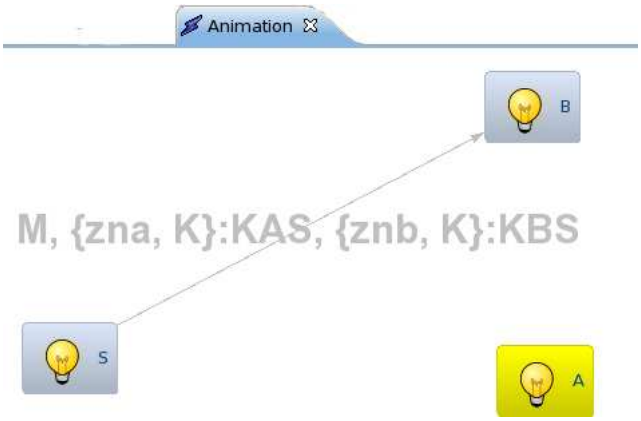


Figure 9: Single step of animation

model correctly describes what they intended. These tools both check the liveness of the protocol and show the output in simple, easy-to-understand formats.

5 Related Work

Work relating to the Elyjah tool can be found in the previously mentioned paper[12]. The most closely related work is the Microsoft Research tool FS2PV[3] which translates F# programs to a low-level π calculus model. This can then be analysed by a theorem prover, ProVerif, to provide security analysis of the implementation. This uses different input and output languages to achieve a similar result to Elyjah. Due to the method of verification used, analysis of protocols beyond a few message exchanges can take a few days. This would make it unsuitable for a development-time tool. The follow-up to this work[2] uses typing rules and drastically reduces the time needed for verification. The potential downside here is the requirement for special refinement types which is in comparison to Elyjah requiring the use of a special API. However F# is not used as much as the Java language that Elyjah is geared towards.

Examples of applications which provide visual representations of cryptographic protocols are ProtoViz[8], GRASP[6], GRACE[5] and TECP[15]. These tools are primarily designed as education packages for teaching students about cryptographic protocols. While LyTE would be more suitable to teaching students about process calculi such as LySa, it could still be used to demonstrate cryptographic protocols by a demonstrator familiar with LySa. Additionally, these tools use their own language for describing a protocol whereas LyTE uses the LySa language so the protocols can be analysed for security flaws, which is an important point when teaching cryptographic protocols.

6 Conclusion

We developed a series of tools to assist users unfamiliar with process calculi utilise the security analysis of the LySatool. With the LySa Editor's parse checker and ability

to check for matching send/receive processes we have made it easier to write or edit LySa models and with the AnaLySa and VisuaLySa we provide tools to translate a LySa process into a simpler model. As a side-effect of this work we have provided a user of the Elyjah tool with the neccessary tools to check that the LySa output of Elyjah is accurate with respect to the Java implementation. We believe this is more useful to the users of Elyjah than a theoretical proof of the correctness of Elyjah which may be hard to understand. Additionally a simple case-by-case proof is more relevant to the user.

Although left out of this paper for reasons of brevity, Elyjah can also generate Meta-LySa which can be edited and analysed by LyTE. Meta-LySa extends LySa by adding further meta information to a protocol model which encodes the scenario of a protocol run. This is important as new attacks may be available when there are different numbers of servers, initiators or responders.

There are avenues for further work in expanding the integration of LySa into the Eclipse IDE. It would be beneficial to the end-user for the output from the LySatool to be integrated into the VisuaLySa so the possible attacks from an attacker are displayed and the confidentiality of message parts should be displayed. It may also be convenient to have a tool that provides the functionality of the tools such as Spi2Java[14] and Millen and Muller's work[10] but using LySa and the API used by Elyjah. Having tools that work in both directions would allow a user to make changes to either the Java or LySa model and still be able to check the security properties.

With Elyjah, we tried to provide the functionality of cryptographic model analysis to a user who had no inclination to learn how to use these formal models. With the LySa Toolkit in Eclipse the intention was to try to create some tools which appeal to those who wish to take the next step and have more understanding of the output of Elyjah and the underlying LySa.

6.1 Acknowledgements

This work would not have been possible without the assistance of my supervisor Stephen Gilmore, the support of the Laboratory for Foundations of Computer Science at the University of Edinburgh and funding provided by the Engineering and Physical Sciences Research Council.

References

[1] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus. pages 195–209. IEEE Computer Society, 2008.

[2] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement Types for Secure Implementations. In *Computer Security Foundations Symposium*, pages 17–32, 2008.

[3] Karthikeyan Bhargavan, Cedric Fournet, Andrew D. Gordon, and Stephen Tse. Verified Interoperable Implementations of Security Protocols. *Proceedings of the Computer Security Foundations Workshop*, 2006:139 – 152, 2006.

[4] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Automatic validation of protocol narration. *Proceedings of 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 126–140, 2003.

[5] G. Cattaneo, A. De Santis, and U. Ferraro Petrillo. Visualization of cryptographic protocols with GRACE. *J. Vis. Lang. Comput.*, 19(2):258–290, 2008.

[6] Michael Collins Wayne Brown Michael Sherman Dino Schweitzer, Leemon Baird. GRASP: A Visualization Tool for Teaching Security Protocols. In *Proceedings from the 10th Colloquium for Information Systems Security Education*, 2006.

[7] Eclipse IDE. http://www.eclipse.org/, 2007. Webpage hosted by the Eclipse Foundation.

[8] Niklas Elmqvist. Protoviz - A Simple Protocol Visualization, April 2004. www.math.chalmers.se/ elm/courses/security/.

[9] Jean Goubault-Larrecq. Towards Producing Formally Checkable Security Proofs, Automatically. pages 224–238. IEEE Computer Society, 2008.

[10] Jonathan Millen and Frederic Muller. Cryptographic Protocol Generation From CAPSL. Technical Report SRI-CSL-01-07, SRI International, December 2001.

[11] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.

[12] Nicholas O'Shea. Using Elyjah to Analyse Java Implementations of Cryptographic Protocols. In *FCS-ARSPA-WITS'08*, pages 211–226, 2008.

[13] A. Pnueli, M. Siegel, and E. Singerman. Translation Validation. pages 151–166. Springer, 1998.

[14] Davide Pozza, Riccardo Sisto, and Luca Durante. Spi2Java: Automatic Cryptographic Protocol Java Code Generation from spi Calculus. In *AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, page 400, Washington, DC, USA, 2004. IEEE Computer Society.

[15] Jelena Zaitseva. TECP Tutorial Environment for Cryptographic Protocols. Master's thesis, Institute of Computer Science, University of Tartu, 2003.

Decision Support for Intrusion Detection Data Collection

Ulf E. Larson[†], Stefan Lindskog[⋆], Dennis K. Nilsson[†], and Erland Jonsson[†]

[†]Department of Computer Science and Engineering,
Chalmers University of Technology, Göteborg, Sweden
{ulf.larson, dennis.nilsson, erland.jonsson}@chalmers.se

[⋆]Centre for Quantifiable Quality of Service in Communication Systems,
Norwegian University of Science and Technology, Trondheim, Norway
stefan.lindskog@q2s.ntnu.no

Abstract

Data collection is a critical but difficult activity for intrusion detection. The amount of resources that must be monitored and the rate at which events are generated makes it impossible to use an exhaustive collection strategy. Furthermore, selection and configuration of data collection mechanisms is a tedious and elaborate task for both designers and operators. Therefore, we propose a decision support system (DSS) for selecting and configuring data collection mechanisms. We suggest a generic system model for selecting data collection mechanisms based on the amount of excess data produced. We also provide an implementation of the system. The DSS reduces effort, time, and expertise required in the selection process, and allows both designers and operators to focus on intrusion detection rather than selection and configuration of data collection mechanisms.

Keywords: intrusion detection, decision support, logging, adaptive security, data collection

1 Introduction

Both intrusion detection system (IDS) designers and operators have a complicated threat picture to assess. External penetrators may attempt to download and install malicious code such as rootkits and backdoors. Insiders may abuse their privileges to gain access to restricted documents. Thus, it is necessary to be able to detect a wide variety of attacks and intrusions. This requires the use of several detection engines and data collection mechanisms for collecting the data required by the detection engines. When a data collection mechanism is enabled, it consumes system resources, e.g., processing power and disk space. It is therefore important that the mechanisms are configured so that they only collect the data that is needed by the detection engines. It is also important that the mechanisms are only active when there is a need for the data that they collect.

However, to properly decide which mechanisms should be active and what configuration they should have is not by any means straightforward. Numerous collection mechanisms exist, all having considerably different characteristics and capabilities of collecting data. Some are specific and tightly coupled to applications [1]. Others are general purpose and support a wide range of configuration options [10]. Furthermore, several data collection mechanisms may collect the same data. Finally, the need for data collection may change over time. This demands consecutive enabling, reconfiguration, and disabling of them, which is a resource-demanding and time-consuming process when performed manually.

Whether caused by complex configuration options or insufficient time, ad-hoc selection and configuration may result in vast amounts of collected data that is not subsequently used. Therefore, in this paper, we propose a decision support system (DSS) which assists IDS designers and operators in making informed decisions regarding mechanism selection and configuration. The system reduces the effort, time, and expertise required for maintaining and tuning a data collection architecture. Furthermore, the system promotes maintaining a resource saving data collection strategy and focusing on reviewing detection alerts instead of selecting and configuring data collection mechanisms.

The contributions of this paper are:

- Identification of required functionality and reference data needed by the DSS. The purpose of the DSS is automatic selection and configuration of data collection mechanisms.
- A conceptual model of the DSS which relates the functionality to the reference data.
- An automated implementation of the DSS for selecting data collection mechanisms based on the amount of excess data produced.

The remainder of this paper is organized as follows. Section 2 discusses related work and Section 3 provides a note on terminology. Section 4 motivates the research and provides a problem statement and in Section 5, the conceptual DSS system model is presented. Section 6 describes an implementation of the system and discusses parts of the design more extensively. Section 7 describes how the system was evaluated and shows screen shots from the operation. Section 8 provides a discussion and outlines future work. Finally, Section 9 concludes the paper.

2 Related Work

Previous work that discusses how data collection can be adjusted to a current need includes GrIDS [14], EMERALD [11], and AAFID [13]. These efforts have focused mainly on scalability issues and transparent enabling and disabling of components. Thus, adjustment has mainly been on the detection algorithm level while the sensor has been treated as a static source of input data. GrIDS uses network sniffers as log source, and to our knowledge they do not perform adaptive collection. EMERALD supports configurable event structures which select parts of the data generated by the log source, which is a step towards adaptive collection. The AAFID system is the system that most closely resembles our effort. However, there is no discussion regarding reconfiguration of the collection mechanism.

Adjustment in relation to intrusion detection has been further explored in [12] where adjustment of resources to the current detection need is investigated. Furthermore, stackguard and the following SAM project [3, 5] also take an adaptive approach but in the context of buffer overflow attack prevention. Two dynamic intrusion detection frameworks are IDIAN [4] and STAT [18]. Both frameworks propose advanced dynamic data collection sensors for intrusion detection. IDIAN mentions resource reduction by means of selection and de-selection of event generators but does not provide any details. Furthermore, in [6] and [19], attack strategy analysis is adapted to on-going attacks to determine where to focus intrusion detection resources. This approach could be complimentary to the approach described in this paper, thus using the DSS to enable and disable collection mechanisms as new detection engines are required.

3 A Note on Terminology

Data collection is the activity of collecting data from an environment, e.g., a computer system or a network. A *data collection mechanism* (or short, mechanism), is an entity that collects data.

The *capability* of the mechanism denotes what data the mechanism can collect. A *mechanism instance* is a process which is currently collecting data. The *configuration* of a mechanism instance determines what data the mechanism currently collects.

A *detection engine* is a component that implements a detection algorithm or detection method that can classify input data as normal or intrusive [2]. Examples are string matching, neural networks, or expert systems. A *detection engine data requirement* is a set of data that a detection engine requires for its detection.

A *request* is a message sent from an operator or designer to the DSS. It contains the name of a detection engine and denotes whether the detection engine should be enabled or disabled. A *decision* is produced by the DSS in response to a request.

4 Decision Support for Data Collection

We present two cases where an IDS designer or operator benefit from using a DSS for selecting data collection mechanisms. These cases are illustrated in Figures 1(a) and 1(b).

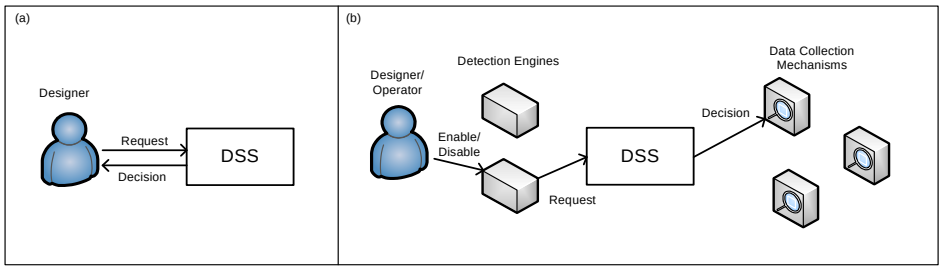


Figure 1: Batch processing of data during IDS framework deployment (a), and continuous processing during operation (b)

Figure 1(a) illustrates the use of the DSS when an initial IDS framework is deployed. In the figure, the designer inputs the detection engine data requirements as a request to the DSS. The DSS responds by outputting its decision regarding selection and configuration of data collection mechanisms.

In Figure 1(b), the DSS is used during IDS operation. In the figure, the operator enables and disables detection engines on demand, e.g., to maintain a resource saving detection baseline and to focus additional resources where they are needed (i.e., adaptive intrusion detection). When a detection engine is enabled or disabled, the DSS is activated to determine which corresponding data collection mechanism should be enabled or disabled. This way, mechanisms are only enabled when they are needed. Additionally, when detection engines are added to or removed from the system, the designer does not need to manually decide which data collection mechanisms that are no longer needed, or what capabilities new mechanisms must have. Thus, using the DSS instead of manually selecting and configuring the data collection mechanisms has the following advantages:

1. The designer is automatically provided with mechanism configuration decisions during deployment. This promotes faster deployment and more resource efficient initial framework layout.

2. When the designer modifies the current framework layout during operation, there is no need to manually decide which data collection mechanisms that are no longer needed, or what capabilities new mechanisms must have. This promotes shorter maintenance while still promoting resource efficient operation.

3. It allows for performing efficient adaptive intrusion detection. By using the DSS as a component in an already established framework, the most resource efficient data collection mechanisms can be enabled automatically when they are needed. This allows the operator to focus on reviewing detection engine output while maintaining a resource efficient operation.

The main problem that is addressed in this paper is how to design a general purpose DSS model for selection of data collection mechanisms. It also addresses how the DSS should be implemented to select and configure data collection mechanisms such that the selected mechanisms collect the data that is required by currently enabled detection engines, while producing as little excess data and using as few mechanism instances as possible.

5 DSS Design

Based on the two cases described in the previous section, we developed the generic conceptual DSS model that is illustrated in Figure 2. This section describes the model and its components. It also describes the flow of operations between the components.

5.1 Conceptual Model

The conceptual model in Figure 2 shows the IDS operator or designer and the DSS. The con-

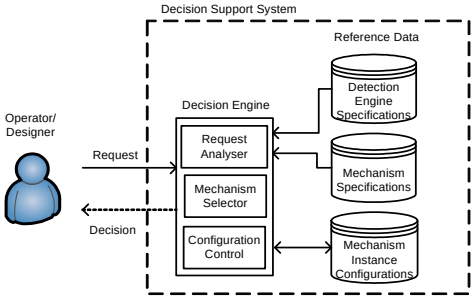


Figure 2: The conceptual DSS model

ceptual model is based on two main concepts: *decision engine* and *reference data*. The decision engine is the operative part that processes requests and makes decisions. The decision engine decides which mechanism, if any, that is best fit for collecting the data required by the detection engine. It also decides how an instance of this mechanism should be configured. The decision engine consists of the three components *Request Analyzer, Mechanism Selector*, and *Configuration Control*. The reference data contains information regarding mechanisms and detection engines and supports the decision engine in making decisions. There are three items of reference data denoted *Detection Engine Specifications, Mechanism Specifications*, and *Mechanism Instance Configurations*.

5.2 Reference Data

Reference data is used by the decision engine during operation. The Detection Engine Specifications contain the detection engine data requirements, e.g., the data that needs to be collected on behalf of the engine. The Mechanism Specifications contain the capabilities of the mechanisms. Both Mechanism Specifications and Detection Engine Specifications are static data and change

only when mechanisms and engines are added to or removed from the IDS framework. Mechanism Instance Configurations contain the current configuration of each mechanism instance. They are dynamic and change when mechanism instance configuration changes.

5.3 Decision Engine Functionality

The functionality of the decision engine is divided among three components: Request Analyzer, Mechanism Selector, and Configuration Control.

5.3.1 Request Analyzer

The purpose of the request analyzer is to inspect requests and determine whether the mechanism selector is required or not. For this purpose, it determines whether a request denotes enabling or disabling of a detection engine, and whether collection of the data in the detection engine data requirement is supported. If the request analyzer decides that data should be collected and that it is possible to collect the data, the mechanism selector is invoked.

5.3.2 Mechanism Selector

The mechanism selector has two tasks. It decides which mechanism is best fit for collecting data, and it decides upon an appropriate configuration for an instance of this mechanism. To determine the best fit mechanism, a strategy for assigning cost to different alternatives is used. Furthermore, the Mechanism Selector ranks eligible alternatives and selects the least costly one. Cost assignment is determined by appropriate algorithms for calculating the amount of excess data collected by a specific mechanism. Thus, the Mechanism Selector decides what mechanism is best fit to collect the data with respect to the detection engine data requirement and decides upon an appropriate configuration for an instance of the best fit mechanism.

5.3.3 Configuration Control

The Configuration Control keeps track on currently active mechanism instances and their configurations. It also keeps track on which mechanism instances are collecting data on behalf of which detection engine(s). This function is necessary since the Mechanism Selector must be able to decide whether it is possible to reconfigure a mechanism instance instead of starting a new instance when deciding on an appropriate configuration. It is also necessary for determining when mechanism instances become unused and can be disabled. During operation, the Configuration Control updates mechanism instance configurations to reflect enabling and disabling of detection engines.

5.4 Flow of Operation

The flow of operation in the decision engine part of the DSS is illustrated in Figure 3.

In the figure, the *Disable detection engine?*, and *Collection supported?* decision objects represents actions performed by the request analyzer. The *Determine best fit mechanism* and *Determine mechanism instance config.* objects represent actions performed by the mechanism selector, and the *Update mechanism instance config.* object represent the action performed by the configuration control. Furthermore, reference data is denoted with MS for mechanism specification, DS for detection engine specification, MC for mechanism instance configurations, and RQ for request.

The operation is initiated when the decision engine receives a request from the operator or designer. The request analyzer then determines whether the request means that a detection

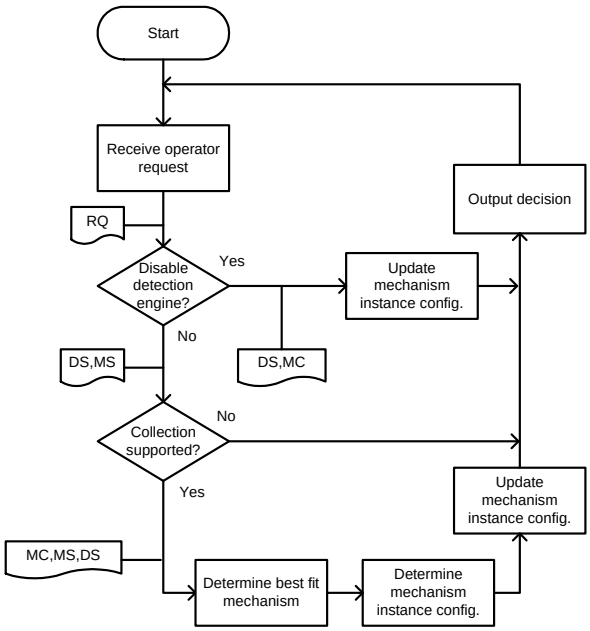


Figure 3: The flow of operation within the decision engine

engine should be disabled or not. If the request denotes disabling, the configuration control updates the mechanism instance configurations accordingly. If the request means enabling, the request analyzer inspects the request to find which detection engine is requested to be enabled.

The request analyzer then uses the detection engine specifications and mechanism specifications to determine whether collection is supported or not. If the request analyzer cannot find any suitable mechanism, it announces that the detection engine data requirement cannot be supported. For this purpose it uses the mechanisms specifications and the detection engine specifications. If the detection engine data requirement is supported, the mechanism selector decides which mechanism is the best fit. For this purpose it uses the detection engine specifications and the mechanism specifications. The mechanism selector then determines whether an already active instance should be reconfigured or if a new instance should be invoked, and it determines the resulting configuration. For this purpose it uses the mechanism instance configurations. Finally, the configuration control updates the mechanism instance configurations.

After having performed the necessary operations the decision engine outputs its decision to the operator. This is either a message that the data cannot be collected, or a message denoting what mechanism is the best fit and how an instance of the mechanism should be configured.

6 DSS Implementation

We made an implementation of the generic conceptual DSS model. The implementation supports both batch and continuous processing and is written in the Perl programming language. The purpose of the implementation is to make decisions with respect to the amount of excess data produced and the number of processes active for collection. This purpose is reflected both in the strategy for selecting and configuring the best fit mechanism, and in the process of identifying reference data content. We decided to use an event-based context for our DSS. Event-based detection is the foundation for popular detection strategies such as, e.g., sequence-based anomaly detection and signature-based misuse detection and was therefore deemed suitable. In

an event-based context, mechanisms react on *events* in a system, e.g., issued system calls or received network packets. Each event is described by one or more *attributes*, e.g., `arg, UID`, or `source IP`, and corresponding *values*, e.g., */etc/passwd, 0*, or *192.168.0.1*.

6.1 Mechanism Selection Strategy

The difference between required and collected data is the cause of excess data. In an event-based context, the difference is determined by two factors.

1. The number of events that are collected in excess to the number of events that are required. Thus, assume that the only events that are required to be collected are the arrival of network packets at port 80. A collection mechanism that collects packets on all ports obviously causes excess data (i.e., all packets not arriving at port 80).

2. The amount of information per event that is collected in excess to what is required. Thus, assume that during system call collection, only the name and the argument for each system call are required. A mechanism that also collects the return value causes excess data (i.e., the return value).

Our strategy is thus to select the mechanism that collects the least amount of events and the least amount of information per event in excess to what is required. Furthermore, we believe that the number of events has a larger impact on excess data than the extra information per event. Thus, we use a two step process where the second step is performed only if more then one mechanism yields an equal and lowest cost in the first step.

6.1.1 Excess Data from Collected Events

Depending on the implementation of a mechanism, it either collects all possible values for specific arguments. For example, it is not possible to configure `strace` [15] to collect those system calls where the return value is `-1`. Other mechanisms, such as `linux audit` [7] can be configured to collect specific values for all of its attributes. For example, it is possible to configure it to collect all system calls where `UID` is `0`.

In the first step of the selection process, the DSS assigns a cost to each attribute in the data requirement where the requirement specifies a specific value, but where the mechanism cannot support collection of individual values. Thus, assume that a data requirement states that the only value that is interesting for the attribute *IP address* is `192.168.0.1`. If a mechanism cannot be configured to collect this address only but rather all addresses, a cost is implied for the IP address attribute. Furthermore, if the mechanism can be configured to collect this address only, there is no cost. The total cost C_{event}, for a mechanism is calculated by performing this test on each attribute (see (1) and (2)). In (1) and (2), m_i and d_i denote the value of the *ith* attribute of the mechanism and data requirement respectively. Furthermore, *ind* denotes that individual values can be collected for the attribute and *all* denotes that individual values cannot be collected.

$$P(m_i, d_i) = \begin{cases} 1, & m_i = all \wedge d_i = ind \\ 0, & otherwise \end{cases} \tag{1}$$

$$C_{event} = \sum_{i=1}^{n} P(m_i, d_i) \tag{2}$$

Thus, the higher the correspondence between requested and collected values, the lower the cost.

6.1.2 Excess Data from Extra Information Per Event

As discussed above, a mechanism can support collection of all values or be configured to collect specific values. In the same manner, some mechanisms, e.g., apache, support arbitrary attribute combinations while others, e.g., Linux audit, collect a static set where all attributes are always collected. Unless arbitrary combinations are supported, attributes have dependencies, e.g., other attributes that are collected at the same time.

The second step in the selection strategy is to calculate the cost of excess data that originates from attribute dependencies. In our implementation, we use Algorithm 1 (see below) to calculate the amount of excess data by counting the number of dependent attributes that are collected together with the required attributes. In Algorithm 1, DR is short for Data Requirement and A is the resulting set of attributes.

Algorithm 1 Algorithm for obtaining number of dependencies for an analysis engine data requirement

```
1: {Main algorithm calcDep}
2: for all attributes a_i ∈ DR do
3:    if a_i not in set A then
4:       Add a_i to A
5:       Get list of dependencies D for a_i
6:       Call addDeps(D)
7:    end if
8: end for
9: {Sub addDeps}
10: for all d_i ∈ D do
11:    if d_i not in A then
12:       Add d_i to A
13:       Get list D of dependencies for d_i
14:       Call addDeps(D)
15:    end if
16: end for
```

We then subtract the number of attributes in the data requirement from the number of attributes in set A to obtain the total cost (C_{dep}). This is shown in (3).

$$C_{dep} = |A| - |DR \cap A| \tag{3}$$

Thus, the smaller amount of attributes that a mechanism collects in excess to what is required, the lower cost it gets.

6.1.3 Mechanism Configuration

Additionally, the DSS must decide whether an active mechanism instance should be reconfigured, or if a new mechanism instance should be invoked. In our implementation, the DSS always suggests a reconfiguration if that means collecting less data, i.e., if the following condition holds: The amount of data collected by the reconfigured mechanism instance is less than or equal to the amount of data collected by the new mechanism instance together with the amount of data collected by the old mechanism instance.

6.2 Identification of Reference Data Content

Reference data is used when selecting the best fit mechanism. Since the selection strategy operates on events, this is reflected in the reference data.

6.2.1 Mechanism Specification Content

The Mechanism Specification describes what capability to collect data the mechanism has and it describes how the mechanism can be configured. We surveyed manual pages and white papers for several common data collection mechanisms to find capabilities and configuration options. From the survey we determined that the specification must contain the following elements: *event type, attributes, values, attribute dependencies, collection restrictions*, and *monitoring scope*. The event type represents, e.g., network connections or issued system calls. The attributes (e.g., `source IP` or `pid`) and values (e.g., `192.168.0.1` or `read`) represent the capabilities of the mechanism to collect data. The capabilities affect the number of events that are collected in excess of the number of required events. The attribute dependencies determine the amount of excess information per collected event. Collection restrictions and monitoring scope are used to describe restrictions of the mechanism which also affect the number of events collected.

Attribute names were derived from terms used in the manual pages and white papers. Certain terms are commonly accepted, which means that names such as pid, UID, source IP, and source port will likely be used in the description of any mechanism that collects this information.

6.2.2 Detection Engine Data Content

The detection engine data requirements can be readily obtained from manual pages, white papers, and research articles. However, instead of surveying several detection engines and creating (possibly subjective) detection engine specifications, we implemented a data requirement generator that produces data requirements. The generator made it easier to test various functionality of the DSS and is further described in Section 7.

6.2.3 Mechanism Instance Configurations

Mechanism instance configurations describe the current configuration of a mechanism instance. The configurations therefore contain event types, attributes, and dependencies from the Mechanism Specifications and values from the data requirements. They also contain identifiers for both the mechanisms and the detection engines.

7 Evaluation

To evaluate the implementation, we first created mechanism specifications for six popular data collection mechanisms: `Linux audit, netstat, strace, ltrace, syscalltracker`, and the `apache mod_log_config` module [7, 9, 15–17]. An excerpt from the strace specification is provided below.

```
<syscall-sname:[a,*]:[sarg,sret] ...  none-[single]>
```

The specification contains the event type (syscall), attribute names (`sname,sarg`), values ([a,*]), attribute dependencies ([sarg,sret]), restrictions (none), and monitoring scope ([single]). 'a' denotes that individual attribute values are supported for this attribute, and '*' denotes that all values can be collected. The restrictions denote whether the mechanism has restrictions, e.g., "port=80" for `apache`, and the monitoring scope denotes whether the mechanism collects data

from one or several processes simultaneously. One <attribute:[value]:[dependency]> triple is defined for each attribute for the event type.

After having created the mechanism specifications, we implemented a data requirement generator using the Perl programming language. The data requirement generator outputs data requirements as input for the DSS. Each data requirement is created by randomly selecting an event type and a set of <attribute:value> pairs. An example of an input data requirement is given below.

```
<syscall-sname=execve:uid=*:euid=0>
```

This particular requirement states that the detection engine operates on system calls and that `execve` system calls for all users having an effective user id of 0 (e.g., root) should be considered. This requirement could, e.g., be issued by a detection engine that attempts to reveal the presence of buffer overflow attacks.

To evaluate and verify the functionality and the implemented DSS strategies, we run the DSS in both batch and continuous mode. Figure 7 illustrates the setup.

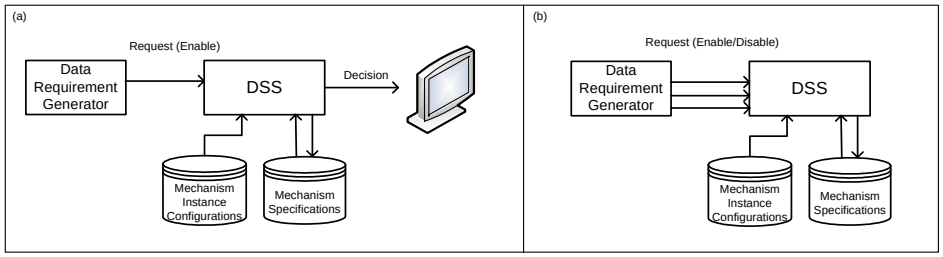


Figure 4: The system setup for evaluation of batch (a) and continuous processing (b)

In batch mode 1(a), a set of input data requirements are input from the generator to the DSS. The DSS selects the combination of mechanisms that are most appropriate for meeting the requirement and outputs this to the screen. In batch mode, all generated mechanism instance specifications are reset before the next run.

In continuous mode 1(b), one requirement at a time is input to the DSS which correspondingly updates the mechanism instance configurations. In this mode, no output is generated to the screen and the mechanism instance configurations are not reset between runs.

7.1 DSS output

In batch mode, the DSS outputs data to the screen. Figures 5 and 6 shows screen shots of input data requirement and corresponding output from the DSS when run in batch mode. In the output listing in Figure 5, the list of attribute dependencies has been abbreviated to save space.

As seen in the output, the DSS has concluded that for the three data requirements, two mechanisms (`apache` and `linux audit`) must be used. Additionally, the output lists the attributes that must be set and the values that should be used for respective attribute. `linux audit` uses a collection strategy where all attributes are collected by default and thus, several attribute dependencies exist. `apache` on the other hand only collects the attributes that are requested and hence, no dependencies exist there. The listing in Figure 6 illustrates the effect of the reconfiguration strategy. As seen in the figure, the DSS requires that two instances of linuxaudit are started. The reason for this is that in the first data requirement, `euid` denotes an individual value (100) and `fsuid` all values (*). In the second requirement, the `euid` denotes

```
syscall-uid=*:euid=*:gid=5:fsuid=1
syscall-uid=*:euid=*:gid=5:fsuid=*
connection-srcIP=192.168.0.1:rproto=*:state=*

#####################################################
            Configuration decision:
#####################################################
Configure one instance of 'apache' as follows:
[srcIP=192.168.0.1][rproto=*],[state=*],
#####################################################
Configure one instance of 'linuxaudit' as follows:
[euid=*],[uid=*],[fsuid=*],[gid=5]
-----------------------------------------------------
Following are dependencies that will also be collected
sret
...
pname
#####################################################
```

Figure 5: Batch output from three data requirements

```
syscall-uid=*:euid=100:gid=5:fsuid=*
syscall-uid=*:euid=*:gid=5:fsuid=50

#####################################################
            Configuration decision:
#####################################################
Configure one instance of 'linuxaudit' as follows:
[euid=100][uid=*],[fsuid=*],[gid=5]
-----------------------------------------------------
Following are dependencies that will also be collected
...
#####################################################
Configure one instance of 'linuxaudit' as follows:
[euid=*],[uid=*],[fsuid=50][gid=5]
-----------------------------------------------------
Following are dependencies that will also be collected
...
#####################################################
```

Figure 6: Reconfiguration versus start new instance decision

all values and `fsuid` an individual value. If the same mechanism instance would serve both requirements, it would trigger on all `euid` and `fsuid` values. This would generate more data than if two individual configurations were used (one collecting only data where euid=100 and one collecting data where fsuid=50).

7.2 Evaluation Results

The evaluation of the mechanism reveals two advantages with using the DSS compared to performing the selection and configuration manually. First, the DSS is consequent in its decision. Manually selecting and configuring the mechanisms might produce different results depending on the allowed time and the skill level of the designer. Second, since the DSS is an automated process, it is considerably faster than the manual process.

8 Discussion and Future Work

The DSS implementation in this paper uses a simple cost assignment scheme (e.g., assigning a unit cost for each extra event attribute collected) for selecting mechanisms. Furthermore, to derive the information in the specifications, we only used statically available information (surveyed white papers and manual pages). More elaborate and exact cost assignment schemes can obviously be devised, taking for example dynamic information in consideration. Dynamic information is generated from a running system and can provide, e.g., the average number of log records over time or the average size of specific log records. However, these estimations demand extensive measurement. They are also dependent on system load and may thus change over time.

The current DSS implementation makes decisions with respect to an estimation of the amount of excess data collected. However, the DSS model is general enough to allow for addressing the more general problem of saving resources. Thus, complementary implementations may be suggested to make decisions regarding other resources as well, such as the impact on performance of the host system. This would imply devising new strategies and functionality for cost estimations, and extending the mechanism specifications with more information.

The main effort related to deploying and operating the DSS is the creation and modification of mechanism and detection engine specifications. Mechanism specifications need to be created only when new data collection mechanisms are installed. Furthermore, if common mechanisms are used, they will likely not change (modifying, e.g., tcpdump would render *many* applications useless). Detection engine specification may or may not be more frequently changed. We believe that when a detection engine changes, it is mainly the algorithm that changes, and not the input data requirement.

Future work includes integrating the DSS in the operator-centric and adaptive intrusion detection architecture proposed in [8]. It also includes evaluation of different techniques for mechanism selection. Additionally, we want to explore cost estimations for, e.g., consumed CPU power.

9 Conclusion

In this paper we have proposed a decision support system (DSS) that assists intrusion detection system (IDS) designers and operators in selecting and configuring data collection mechanisms. The DSS is applicable both during deployment and operation of an IDS framework. We have provided a conceptual model of the system including required functionality and reference data. We have also provided a simple test implementation of the system in which the selection of mechanisms is based on the amount of excess produced data. It is demonstrated that the system reduces effort, time, and expertise required by IDS designers and operators.

Acknowledgment

The work at Chalmers University of Technology is financially supported by the Swedish Emergency Management Agency. The work at the Norwegian University of Science and Technology is financially supported by the research council of Norway.

References

[1] Apache mod_log_config module. `http://httpd.apache.org/docs/2.2/mod/mod_log_config.html`. Visited August 9, 2008.

[2] E. Lundin Barse. *Logging for Intrusion and Fraud Detection*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2004.

[3] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*, pages 63–78, San Antonio, TX, USA, January 26–28, 1998. USENIX Association.

[4] R. Feiertag, S. Rho, L. Benzinger, S. Wu, T. Redmond, C. Zhang, K. Levitt, D. Peticolas, M. Heckman, S. Staniford, and J. McAlerney. Intrusion detection inter-component adaptive negotiation. *Computer Networks*, 34(4):605–621, 2000.

[5] H. M. Hinton, C. Cowan, L. M. L. Delcambre, and S. Bowers. SAM: Security adaptation manager. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC 1999)*, pages 361–370, Phoenix, AZ, USA, December 6–10, 1999. IEEE.

[6] M-Y. Huang and T. M. Wicks. A large-scale distributed intrusion detection framework based on attack strategy analysis. In *Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium, September 14–16, 1998.

[7] *Linux Audit Subsystem Design Documentation for Kernel 2.6*, 2004. Visited August 9, 2008.

[8] U. E. Larson, S. Lindskog, D. K. Nilsson, and E. Jonsson. Operator-centric and adaptive intrusion detection. In *Proceedings of the Fourth International Conference on Information Assurance and Security (IAS'08)*, pages 161–166, Naples, Italy, September, 8–10, 2008. IEEE.

[9] ltrace – default branch. `http://freshmeat.net/projects/ltrace`. Visited August 9, 2008.

[10] Sun Microsystems. *SunSHIELD Basic Security Module Guide*. Mountain View, CA, USA, 1995.

[11] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, MD, USA, September 7–11, 1997.

[12] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response systems. In *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2344–2349, Nashville, TN, USA, October 8–11, 2000. IEEE.

[13] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 26:547–570, September 2000.

[14] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, Baltimore, MD, USA, October 22–25, 1996.

[15] strace – default branch. `http://sourceforge.net/projects/strace`. Visited August 9, 2008.

[16] syscalltrack. `http://syscalltrack.sourceforge.net`. Visited August 9, 2008.

[17] Netstat systems. Netstat. `http://www.netstat.net/`. Visited July 15, 2008.

[18] G. Vigna, R. A. Kemmerer, and P. Blix. Designing a web of highly-configurable intrusion detection sensors. In *Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 69–84, Davis, CA, USA, October 10–12, 2001.

[19] J. Yuill, S. F. Wu, F. Gong, and M-Y. Huang. Intrusion detection for an on-going attack. In *Proceedings of the Second International Workshop on the Recent Advances in Intrusion Detection*, West Lafayette, IN, USA, September 7–9 1999.

Pareto-Optimal Architecture according to Assurance Indicators*

Frank Innerhofer-Oberperfler
University of Innsbruck, Austria
frank.innerhofer-oberperfler@uibk.ac.at

Fabio Massacci, Artsiom Yautsiukhin†
Univesity of Trento, Italy
{Fabio.Massacci,evtiukhi}@dit.unitn.it

Abstract

In this paper we present an approach and algorithm for selecting the "best" secure architecture for supporting a business process according to a variety of assurance indicators. The key difficulty is to select an architectural design in presence of multiple indicators that might offer alternative notions of minimality. Therefore we must use the notion of Pareto optimality in order to select alternatives that are not dominated by others.

1 Introduction

The financial and management scandals of the last few years have spurred legislators and regulators of different countries to take initiative to protect customers and shareholders. While legislators do not mandate how business should be conducted, they require that companies should show that "they are in control of their business" and that such control is correctly reported to interested stakeholders [7].

Since most business processes are automated or rely heavily on an IT infrastructure such business controls are in practice very often controls embedded in information systems or controls supported by IT. Companies are therefore required to show that they have in place the necessary IT controls and that they have ways to assess their control system and control levels.

According to COBIT [7, pag.9]:

> Enterprises need an objective measure of where they are and where improvement is required, and they need to implement a management tool kit to monitor this improvement. Figure1 shows some traditional questions and the management information tools used to find the responses, but these dashboards need indicators, scorecards need measures and benchmarking needs a scale for comparison.

Obviously the ideal notion of indicator for a manager is the dashboard with red, green and yellow. Unfortunately life is more complex and often we must make decisions considering a variety of indicators. We must judge how the compliance goals of a company are reached by considering multiple and possibly conflicting indicators.

Management guidelines such as CoBIT describe how to set control goals (for security and assurance) and provide guidance in the definition of indicators. But such guidelines do not specify how one can use indicators of individual control goals to assess the overall enterprise architecture. In our previous paper [5] we have outlined how a security analyst can evaluate the impacts of security breaches on business objectives of an enterprise. In [16, 15] we provided an approach for aggregating security indicators for a business process (BP) and for selecting the most secure process model. However, both methods have a limitation: the use of a *single* indicator.

*This work was partly supported by the EU-IST-FP6-IP-SERENITY and IST-FP6-IP-SENSORIA projects.

†Contacting Author: `evtiukhi@disi.unitn.it`

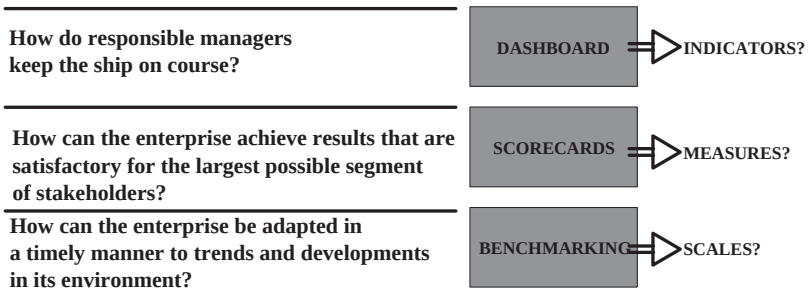


Figure 1: Management Information - From CoBIT

1.1 Contribution of this paper

In this paper we combine both models and consider the case of an evaluation against *multiple-indicators*, which allows an analyst to consider several security criteria at once. We also provide an algorithm for the aggregation of indicators and for choosing the "best" security architecture. In this case however we need to have a criterion for optimality that does account for possibly conflicting criteria. A practical example could be the consideration of indicators for compliance with European privacy directives and US Surveillance laws, which might be conflicting and negatively related.

To this extent we have have chosen the notion of Pareto-optimality: all indicators of a best solution should not be dominated by another solution. This means that we no longer have "the" best solution, but only "a" best solution. All other best solutions improve one indicator at the expense of some other indicator. The final tradeoff between the incomparable alternatives might then be evaluated using different business criteria. In practice an EU company doing business in the US might choose a different solution than a US company doing business in the EU. Still we want to be sure that there are no solutions which can better accommodate both compliance indicators.

This paper is structured as follows: By using an example case (Section 2) we outline and present the underlying Enterprise Model (Section 3). In Section 4 a mathematical model based on the underlying models is presented and followed by an algorithm for assessing the best available solution from a security point of view (Section 5). Finally, we outline some related work in Section 6 and give a conclusion and an outlook.

2 Running Example

Example 1 *In the paper we use our usual loan processing example. Consider a bank holding company which outsources the concrete loan processing to a semi-independent subsidiary. In order to provide the service, e.g., the loan originating process, the subsidiary needs to design the BP, find outsourcing partners and allocate the information systems (applications, servers, databases, etc.) that support the BP. There are a number of design alternatives to choose from: various parts of the BP may be fulfilled in different ways, several outsourcing partners are available which offer the same functionality but provide different protection and trust levels, and last but not least applications and servers are also subjects for various design alternatives. Before providing the service the subsidiary must consider these different alternatives and choose the one which best fits the business needs.*

The holding company is aware of a huge number of risks and losses caused by frauds[1]*. Therefore the holding wants to be sure that it is well protected against this type of losses. That is why one of the main criteria for choosing one of the design alternatives for the subsidiary is a low risk of possible frauds.*

[1] in average organizations loose 5% of annual revenue to frauds and abuses and that for banking companies, in particular, median losses are 258 000$ per company [1]

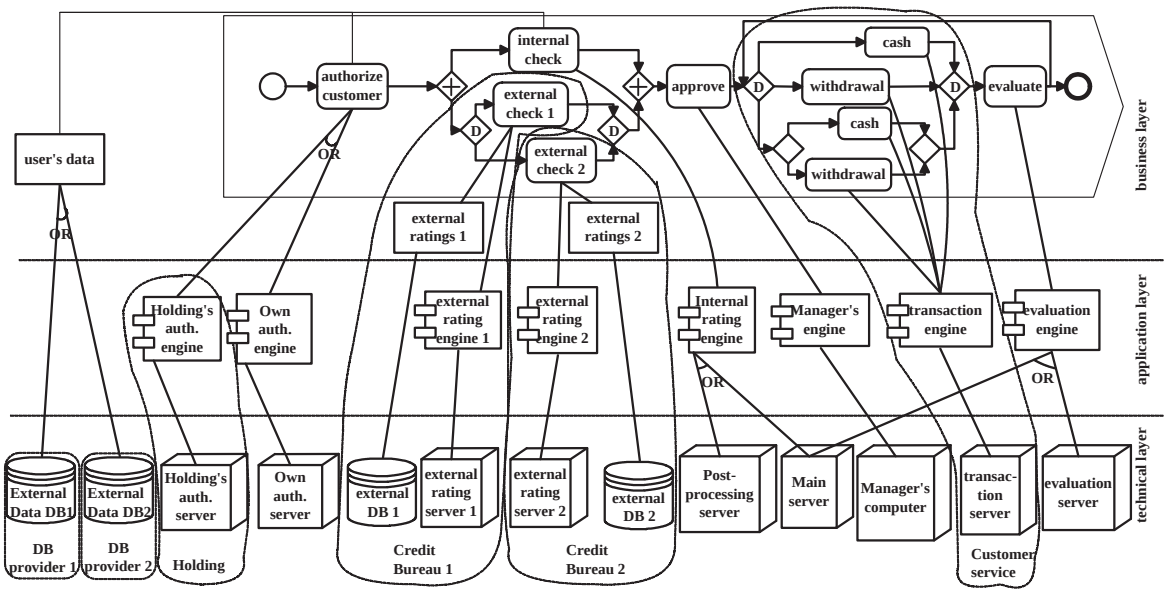


Figure 2: Architecture for loan origination business process.

For the BP modelling we use BPMN (Business Process Management Notation) [20] which is a widely used notation. We added just a small modification: the *design choice* gateway (diamond with letter D inside) to denote an alternative sequence of actions. After the selection only one path (alternative) should be left, i.e., all design choice gateways disappear and we receive a standard conformant BPMN diagram.

Example 2 *For our running example the following – reduced and schematic – model has been determined (Figure 2, upper right corner).*

In the Figure 2 one may see four main parts of the process: AUTHORIZE CLIENT, CHECK TRUSTWORTHINESS RATING, APPROVE *(and finalize) the loan and* REPAYMENT. CHECK TRUSTWORTHINESS RATING *consists of two activities fulfilled in parallel:* INTERNAL CHECK, *done by the subsidiary itself, and* EXTERNAL CHECK, *outsourced to one of two available Credit Bureaus.* REPAYMENT *is an iterative sub-process. Every month a customer pays the agreed amount and the transaction is* EVALUATED. *There are three possible ways of payment: payment by* CASH, *automatical* WITHDRAWAL *from the client's account or the possibility for the client to choose how to pay.*

3 Enterprise Model

For our assessment we need a business oriented model of an enterprise which will help us to separate the different sources of threats on the one hand and provides a way to aggregate the data received from the sources on the other one. Typical sources for data in security systems are reports about security incidents derived from logs. Before using these data we should filter only the events which are relevant for our target of evaluation, i.e., the chosen BP.

In our previous paper we presented the security Enterprise Model based on ideas from [4, 12]. This model allowed us to aggregate security indicators, but did not provide a possibility to make a decision about the best enterprise architecture if several alternatives were possible. Also this model considered filtering of the events relevant for the chosen BP implicitly while in this work we make use the structure of the BP to make the filtering more explicit. In this paper we just outline the main concepts of the enterprise modelling which are relevant for the current paper.

The enterprise meta-model contains three layers (Figure 3):

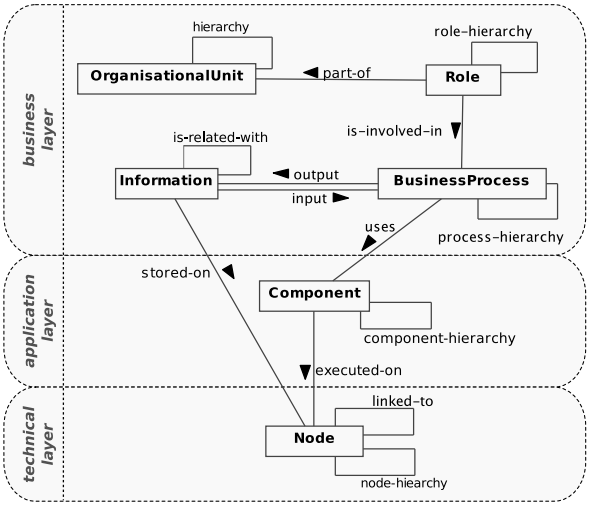


Figure 3: Enterprise meta-model

- **Business Layer.** The business layer contains business oriented artifacts. *Business process* is a predefined sequence of activities leading to the accomplishment of a goal. *Information objects* depict the information which is processed by business processes. The model contains also organizational units and roles, which are not relevant for this paper since we focus only on incidents impacting BP.
- **Application Layer.** On the application layer we have Components. *Components* are the information systems and applications used by some BP (by one of its activities).
- **Technical Layer.** The technical layer contains nodes. *Node* is a software and hardware set which provide required services for components to operate or information objects to be stored. For example, node can be a server with installed Windows Server 2003 and Oracle database.

Using this meta-model we can create an Enterprise Model which can be seen as a tree starting with a chosen BP and containing all elements connected to the BP through a functional dependency relationship.

This Enterprise Model presents in detail all artifacts affecting the chosen BP. The model also explicitly shows the sources of information about security violations related to the artifact.

Example 3 *There are several architectural design alternatives which must be considered in the example. The first one is two possible data bases where* USER'S DATA *can be stored. Another decision to be made is whether to use the holding's authentication system (both application and the server) or to create an own ad hoc solution. The* EVALUATION ENGINE *can be run on the (well protected)* MAIN SERVER *of the subsidiary or on a separate* EVALUATION SERVER.

In the example we have a number of outsourcing relations. The subsidiary decides that it is more profitable to store private user data on an external database: EXTERNAL DATA DB1 *or* EXTERNAL DATA DB2. *The whole* EXTERNAL RATING *service (managing the activity, application and the node) is outsourced to one of two available* CREDIT BUREAUS. *In contrast to the outsouring the* EXTERNAL RATING, *outsourcing of* AUTHENTICATION *to the holding company (application and server) does not include the activity itself because it is the subsidiary which is responsible for performing all required operations (e.g., collect authorization data). Also the* PAYMENT *services are entirely provided by other subsidiaries of the holding which are spread around the city and easily available for customers.*

Note, that the Enterprise Model captures the hierarchy of a BP (even to the level of its atomic activities) but it does not provide information about the control flow of the process. Contributions of

activities to the chosen BP depend on the control flow and we need a more explicit way (a Business Process model) to model these contributions.

3.1 Security requirements in the models

Enterprise and BP models are used for the elicitation and specification of security requirements for the modelled elements. The elicitation starts with the identification of business security objectives (BSOs) (or business security goals). The objectives are made specific for the chosen BP and become security requirements. Then for each level of hierarchy of the BP the requirements are specified for each activity at this level. By hierarchy, here and in the sequel, we mean hierarchy determined by structural activities of the BP ("sequence", "parallel", "loop", "choice"). Such a hierarchy can be easily obtained from a non-hierarchical representation of the BP.

In this paper we consider that there is only one requirement corresponding to one business objective. In other words, each element (activity, node or application) in the model has as many security requirements as many security objectives have been defined in the beginning (in contrast to [5]).

When the lowest hierarchy level of the BP has been reached (all atomic activities) we get the starting requirements in the Enterprise Model. The decomposition of requirements continues according to the Enterprise Model (similar to [5]): first requirements for information objects and components are identified whose violation could impact the previously identified requirements for the activities. Then, security requirements for nodes are specified. At the end, the threats which impact the requirements are identified. It is naturally to identify several threats for the same requirement for a model element, in contrast to identification of requirements where only one requirement for an element is identified for one security objective. These threats may be the cause of violations of any requirement for an element in the Enterprise Model.

1. Incorrect fulfilment of duties by employees and managers violates requirements on the business layer.

2. The threats impacting applications required for the activities violate requirements for components. Internal administration staff (also former administrators) are more probable potential attackers at this level. Skilled outsiders, which can find and exploit vulnerabilities by themselves, also should be considered.

3. The threats impacting physical security and system software required for correct operation of nodes (e.g., operating system, database, etc.) are identified on the physical layer. At this level outsiders are the main attackers, though also insiders should not be neglected.

Example 4 *The general requirement for the BP is the* prevention of fraud*. Going down through the hierarchy of requirements we determine the threats for activities (e.g.,* Rating manipulations *for* INTERNAL RATING *activity,* commit fraudulent transaction *for payment by* CASH*), data (e.g.,* unauthorized modification *of* USER'S DATA*), components (e.g.,* corrupt rating engine*), nodes (e.g.,* capture control *over* RATING SERVER*). The following threats are identified for nodes:* worms, viruses, hacker attacks *(attacks using various exploits),* Trojan horses*. Components can be affected by back doors,* phishing *and* password guessing *attacks on the components requiring authentication and* advance hacker attacks *with home-made application-specific exploits. The business elements can be affected by a dishonest employee* abusing his power *(e.g., not performing internal rating check or deliberately modifying data).*

Note, that though, in theory, some requirements can be connected with themselves (i.e., there are circles) these situations are not very practical and are not considered in this paper. On the business level the circle could be formed by "input" and "output" dependencies (see Figure 2) between Business Processes and Information Objects. Although a situation in which a BP compromises an information object and then, processing the information object compromises itself, is possible, we consider such a situation as improbable. A circle may be also formed by a link between several nodes (e.g., network

connection) but data for our analysis (Section 5) is more likely to be taken from security logs of the nodes and thus we *already* have the total number of attacks for the node.

4 Protection Appraisal DAG

In order to estimate the assurance of a BP we need a suitable data structure derived from the Enterprise and BP models. The mathematical structure we use for our approach is Protection Appraisal DAG (Directed Acyclic Graph). In [16] we described the process in details and provided algorithms for building the Protection Appraisal DAG using a BP as well as for the reconstruction of the BP. In this work we extend the model on the architecture of the enterprise.

Initially, a Protection Appraisal DAG ($\mathcal{PAD}$) is built from a BP specified in the extended BPMN (as it is done in Figure 2). In short, for each activity we add a node (called appraisal node) which denotes the security requirement for this activity. The top node is a general node corresponding to the BP as a whole. The appraisal nodes which correspond to the activities of sub-processes (source set) are connected to the (target) appraisal node for the decomposed activity with a decomposition edge. If we have several alternatives (several sub-processes) to fulfill the same activity we draw several decomposition edges leading to the same target node starting from different source sets. In case an activity is outsourced we add an additional node and connect it with the appraisal node for the outsourced activity. This is done because the expected values of assurance indicators depend on the trust levels of the partners.

A similar strategy holds for converting the Enterprise Model. All enterprise elements are represented with appraisal nodes. Several appraisal nodes (source nodes) are connected with a target node with one decomposition edge if the elements corresponding to the source nodes are required for operation of the element corresponding to the target node.

Example 5 *In Figure 4 we show the part of the Protection Appraisal DAG built for the* RATING CHECK *sub-process. The nodes are colourized differently according to their type of enterprise model element to enhance clarity. To clarify the mapping between Figure 2 and Figure 4 we marked nodes by abbreviating their names (e.g., the nodes corresponding to* INTERNAL RATING ENGINE *are marked as* IRE*). All nodes denoting outsourcing relations have suffix "-O", e.g.,* EXTERNAL CHECK 1 *and* 2 *are denoted by EC1-O, EC2-O. We did not illustrate all threats but only those relevant for* INTERNAL RATING *activity,* INTERNAL RATING ENGINE *and two servers where the engine may run. Other threats should be attached similarly. Note, that the threats are connected with the target node directly for enterprise nodes only (e.g., for* MAIN SERVER *(MS)) while for other elements (impacted also by lower elements) the contributions of the threats are attached to the hyperedge leading to the target node (e.g., for* INTERNAL RATING ENGINE *(IRE)).*

In general each source node in the Protection Appraisal DAG contributes differently to the appraisal of a target node. The different contributions can be captured by assigning several weights to each decomposition edge[2]. For the sake of simplicity, we sometimes just say "assign weights" to an edge when we mean that several sets of weights should be assigned to the edge. Each weight has its own meaning depending on the dependency it belongs to.

Definition 1 *A* Protection Appraisal DAG *($\mathcal{PAD}$) is a quadruple $\langle Q, E, F_e, L_e \rangle$ where Q is a set of appraisal nodes of a BP and an Enterprise Model and E is a set of decomposition edges. Each decomposition edge is an ordered pair $\langle S_q, q \rangle$ from an arbitrary nonempty set $S_q \subseteq Q$ (*source set*) to a single node $q \in Q$ (target node). L_e is a set of edge-dependant labelling functions which assign a set of vectors of weights to each edge. F_e is a set of edge-dependant propagation functions which compute the indicators for a target node taking as arguments indicators for source nodes.*

[2] In fact, we have to use a hypergraph-like structure (called FD-graph) if we want to attach several weights to one edge, but this is not important for the purpose of this paper.

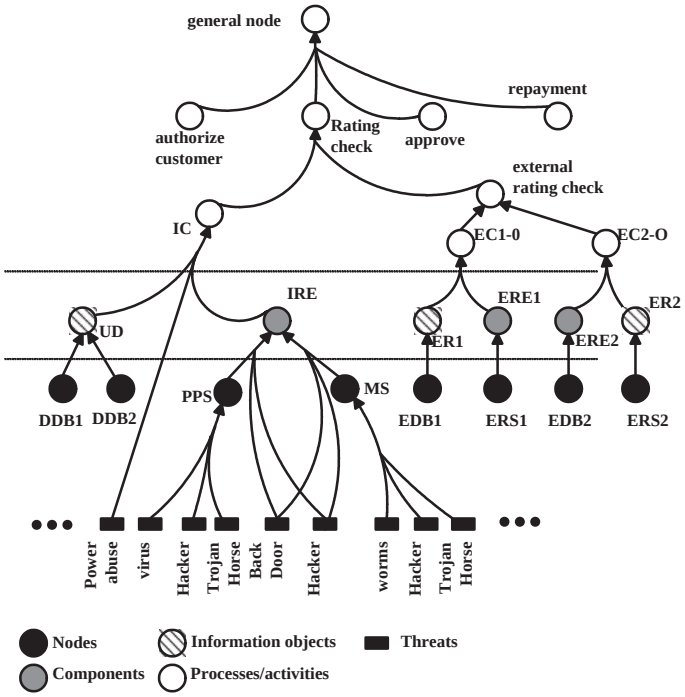


Figure 4: Protection Appraisal DAG.

Relation	**Formula**	**Meaning**
$flow$, $sequence$	$w_i = t_{q_i}/t_q$	Relative time of execution. t_{q_i} - average time for execution of an activity i; t_q - average time of execution of the target activity (whole sub-process)
$choice$	$w_i = p_i * t_{q_i}/t_q$	Relative time of execution multiplied by the probability to choose branch i
$loop$	$w = 1$	No change of the indicator.
$outsourcing$	$w = 1/T_p$	T_p - trust level of partner p.

Table 1: Weights for structural activities

A classical problem in the graph theory is finding the "shortest", i.e. optimal, path. In our approach the "shortest" hyperpath determines the architecture with the highest assurance (with the best values of the assurance indicators). In our previous work [15] we proposed a polynomial algorithm for finding the shortest path using a monotone function as an aggregation function. This algorithm works with only one requirement/indicator. In particular we used a weighted function: The function for an edge $e = \langle S, q \rangle$:

$$F_e = \sum_{\forall q_i \in S} w_i * I_{q_i} \tag{1}$$

where I_{q_i} means the value of indicators for the node q_i belonging to the source set S; w_i is a corresponding weight for the edge. In this work we also use the same weighted function applied for each requirement separately. The weights used for the aggregation function for a BP have the following meanings presented in Table 1.

For all enterprise elements a weight may be seen as the multiplication of two probabilities. Thus the weight for an edge e for a node i and a requirement j will be:

$$w_j^i = p_{req,j} * p_{el}^i;$$

where $p_{req,j}$ - the probability to impact the considered requirement and p^i_{el} - the probability that the considered element will be impacted (but not another one). p^i_{el} may be seen in most cases as a relative time of execution of a higher layer element element using a lower layer one (e.g., the relative time of execution of INTERNAL CHECK on MAIN SERVER) which can be taken from business process descriptions.

Many of the components can be reused. p^i_{el} component is the same for all components of the same vector of weights. $p_{req,j}$ is domain independent and can be used for all vectors on the same level for the same requirement. p^i_{el} component for all threats is equal to 1.

5 Assessment

After creation of the Protection Appraisal DAG the analyst identifies the values of leaf appraisal nodes, i.e., the values of assurance indicators for all threats. The data are received from history records or estimated by security experts. Note that, monitoring events on the business layer require trusted logging procedures in order to have reliable logs. If the activity is outsourced to a subcontractor the values are taken from the corresponding contract.

Now we have a classical problem of finding the "shortest" path: the root set Q' is a set of all leaf appraisal nodes (corresponding to all possible threats) and the target is the top node, which is the general node denoting the quality of protection for the whole process. Our algorithm presented below searches for a multi-objective shortest path (and values) with non-superior functions in a hypergraph using pareto-optimal principle. Unfortunately, the multi-objective optimal path problem is not polynomial, though the algorithm may be significantly optimized [9]. On the other hand, we claim that the algorithm proposed below is polynomial in depth and in number of paths because it is based on the our algorithm from [15].

The main problem with multi-objective cases is that we cannot make an unambiguous decision while comparing two vectors of values (value vector, in the sequel).

Example 6 *Imagine that we evaluate two security objectives:* prevent fraud *and* keep the loan process confidential. *We cannot make an optimal decision while choosing the best host for the* INTERNAL RATING ENGINE *if it is known that: (i) the* POST-PROCESSING SERVER *is more probable to be used for corrupting the* INTERNAL RATING ENGINE *than the* MAIN SERVER, *but (ii) it is less used for compromising confidentiality of the results of the rating check rather than the* MAIN SERVER.

The best thing we can do is to use the Pareto optimality principle to compare vectors. We should compare all elements in two vectors one by one. If *all* values in one value vector are bigger than in the other one we can make an unambiguous decision eliminating the former vector from further consideration. Note, that such decision can be made only if the aggregation function is monotone in all its variables (e.g., weighted function). If at least one element in the first vector is less than the corresponding element in the second one then we should propagate both vectors. These vectors are called non-dominated, because non of them dominates others.

Below we present the algorithm for computing all non-dominated paths. The algorithm is based on [15] and adapted for multi-objective optimal shortest path problem using [14].

In the algorithm by indicator we mean the triple: $i = \langle V, e, 2^I \rangle,\ i \in I$ where V is value vector, e - the last traversed edge, and 2^I is a set of indicators of source nodes of the edge e which were used for calculation of the value vector V. We also assume that all nodes are marked with numbers of edges leading to them and all edges are marked with number of source nodes.

The main Algorithm 1 works similar to the one presented in [15] but the calculation of indicators and their comparison should be specified for a multi-requirement analysis.

We will get several possible indicators by computing value vectors (Algorithm 2). The number of possible indicators we get is equal to the multiplication of the number of indicators for the source nodes. In order to store the path each indicator stores the last edge and all indicators used for calculation. The indicators must be checked for dominance and all dominated indicators must be removed (Algorithm 3).

Algorithm 1 Optimal multi-objective hyperpath

Require: $\mathcal{PAD} = \langle Q, E, F_e, L_e \rangle$: Protection Appraisal DAG;
I_{Leaf} : set of indicators for leaf threats;
Ensure: I : real; set of indicators for all nodes
1: Assign maximum indicators to appraisal nodes;
2: Assign premise indicators (I_{Leaf}) to leaf appraisal nodes;
3: Add leaf appraisal node to a working set;
4: **while** working set is not empty **do**
5: Take randomly a node (q') from the working set;
6: **for** each outgoing edge from q' ($\langle S_q, q \rangle \; . \; q' \in S_q$) **do**
7: Mark the node as visited;
8: **if** All source nodes from S_q are visited **then**
9: Compute Indicators($\langle S_q, q \rangle$);
10: Mark the edge as traversed;
11: **if** all edges leading to q are traversed **then**
12: Choose Non-Dominated Indicators(q);
13: Add q to the working set;

Algorithm 2 Compute Indicators

Require: $\langle S_q, q \rangle$ edge
Ensure: $I(q)$ set of indicators for node q
{calculate all possible sets of indicators}
1: **for** all possible combinations of indicators of source nodes **do**
2: Compute the value vector for an indicator;
{store the hyperpath for the indicator}
3: Assign $\langle S_q, q \rangle$ and used indicators of source nodes to the indicator.

The algorithm aggregates the contributions caused by identified threats and selects the branches which have the non-dominated security indicators. The results of the run of the algorithm is twofold: (i) we receive the set of non-dominated security vectors for the architecture as a whole; (ii) the set of optimal hyperpaths which indicate the most secure system architectures.

At the end of propagation we can compare all non-dominated vectors using ALE analysis [10] and choose the best one, and the best hyperpath correspondingly.

6 Related works

A business oriented perspective on security has long been argued for by many authors in the field [11, 22]. Combining model-based approaches with risk and security methods [13, 19] has in this regard been a path followed by a variety of researchers and practitioners in the last years.

An academic approach that is following a model-based risk analysis is CORAS [3]. CORAS approach uses models mainly for descriptive purposes and to visualize and communicate security aspects to various stakeholders. In contrast we use models to depict dependencies and enhance them with a mathematical model to assess the overall security of the architecture.

Suh and Han [23] use a business model to identify business functions in order to evaluate the relative importance of information assets for these functions. Suh and Han focus solely on the security requirement of operational continuity. For this purpose they use a measure to denote the relative importance of technical assets for business functions.

Morali et al. are using a model-based approach assess confidentiality [18]. Their approach is also based on a model of the architecture and the aggregation of values along the identified dependencies. Our work is different with regard to the weights assigned to these dependencies.

Jürjens [8] has developed the UMLSec approach for model-based security engineering based on UML. UMLSec is an extension of UML that can be used to express security relevant information in UML

Algorithm 3 Choose Non-Dominated Indicators

Require: q - evaluated node
Ensure: $I(q)$ - Indicators for q
{remove all dominated indicators}
1: **for** all indicators of q **do**
2: **if** indicator is dominated **then**
3: Remove the indicator

diagrams of a system. The approach is mainly targeted towards secure system development while we use a enterprise modeling to analyse functional dependencies between business and technical artifacts.

From a security evaluation perspective our work is close to analysis of attack trees. In attack trees the main goal of an attacker (can be seen as violation of the general requirement) is hierarchically decomposed on more concrete steps, fulfilment of which could lead to accomplishing the general goal [21, 17]. Using Enterprise and BP models for construction of Protection Appraisal DAG make goal decomposition more objective. We also provide a quantitative analysis which is based on a well-known risk assessment technique.

In contrast to [2] we first aggregate values (number of attacks in a period of time, which is similar to Annual Rate of Occurrences) and then estimate the impact for business security objectives rather than consider threats separately without linking them with the objectives.

Clark et. al. [6] also aggregated risk levels using an attack tree in order to evaluate the impact on enterprise's mission (business goal), but for aggregation the authors used number of existing vulnerabilities on leaf nodes treating them as risk measures which differs from classical Risk Assessment. Also none of the attack tree based evaluation approaches helps to choose the more secure architectural design.

7 Conclusion

In this paper we presented the approach which outlines how using Enterprise and BP models can help to select the most secure architecture for a BP among several alternatives. The approach also takes into account that some parts of the process or architecture may be outsourced to external partners which may have different trust levels. The analysis considers evaluation using several requirements at once. We supported our approach with algorithms for the aggregation of indicators and selecting the non-dominated architectures. The analysis eliminates all dominated alternatives and returns only the architectures which cannot be easily compared. For such alternatives we proposed to use the well-known ALE analysis to make an unambiguous decision.

References

[1] ACFE. *The 2006 Report to the Nation.* Association of Certified Fraud Examiners, 2006. available via `http://www.acfe.com/documents/2006-rttn.pdf`.

[2] S. Bistarelli, F. Fioravanti, and P. Peretti. Defense trees for economic evaluation of security investments. In *Proc. ARES*, 2006. IEEE Computer Society.

[3] F. den Braber et al., Model-based security analysis in seven steps — a guided tour to the CORAS method. *BT Technology Journal*, 25(1):101–117, 2007.

[4] R. Breu and F. Innerhofer-Oberperfler. Model based business driven IT security analysis. In *Proc. SREIS*, August 2005.

[5] R. Breu, F. Innerhofer-Oberperfler, and A. Yautsiukhin. Quantitative assessment of enterprise security system. In *Proc. WPA*. 2008.

[6] K. Clark, J. Dawkins, and J. Hale. Security risk metrics: Fusing enterprise objectives and vulnerabilities. In *Proc. IAW*, 2005.

[7] IT Governance Institute *Control Objectives for Information and related Technology (COBIT)* Version 4.1. 2007. available via `www.isaca.org/cobit/`.

[8] Jürjens, J. Model-Based Security Engineering with UML *FOSAD 04/05*, Springer, 42-77, 2005.

[9] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000.

[10] L. A. Gordon and M. P. Loeb. *Managing Cybersecurity Resources: a Cost-Benefit Analysis*, 2006.

[11] G. Herrmann and G. Pernul. Viewing Business Process Security from Different Perspectives. In *Proc. of 11th Int.l Bled Electronic Commerce Conf.*, 1998.

[12] F. Innerhofer-Oberperfler and R. Breu. Using an enterprise architecture for IT risk management. In *ISSA*, 2006.

[13] S. A. Kokolakis, A. J. Demopoulos, and E. A. Kiountouzis. The use of business process modelling in information systems security analysis and design. *Information Management & Computer Security*, 8(3):107–116, 2000.

[14] E.Q.V. Martins and J.L.E. dos Santos. The labeling algorithm for the multiobjective shortest path problem. Technical Report 99/005, CISUC, 1999.

[15] F. Massacci and A. Yautsiukhin. An algorithm for the appraisal of assurance indicators for complex business processe. In *Proc. QoP.*, 2007.

[16] F. Massacci and A. Yautsiukhin. Modelling of quality of protection in outsourced business processes. In *Proc. IAS*, 2007.

[17] S. Mauw and M. Oostdijk. Foundations of attack trees. In *Proc. CISC*, 2005.

[18] A. Morali et. al., IT Confidentiality Risk Assessment for an Architecture-Based Approach. *Proc. BDIM*, 2008.

[19] M. zur Muehlen and M. Rosemann. Integrating Risks in Business Process Models. *Proc. ACIS*, 2005.

[20] OMG (Object Management Group). *Business Process Modeling Notation Specification*, 1.0, 2006. available online.

[21] B. Schneier. Attack trees: Modelling security threats. *Dr. Dobb's journal*, December 1999.

[22] B.v. Solms and R.v. Solms. From information security to...business security? *Computers & Security*, 24(4):271–273, June 2005.

[23] B. Suh and I. Han. The IS risk analysis based on a business model. *Information & Management*, 41(2):149–158, 2003.

Towards Access-Control Policies Written By Managers (Invited Talk)

Michael Huth
Imperial College London
mrh@doc.ic.ac.uk

Today most access-control policies reside within IT systems—rule files of firewalls being an example thereof. Thus policy writers are mostly technically skilled people. Tomorrow more policies will have to be formulated by managers or owners of assets that are not directly represented or identifiable within IT systems—legal audit requirements and obligations in work flows being two examples. This creates a need for policy languages that can be used and analyzed by people with little or no exposure to information technology.

We design a policy language that goes some way towards this strategic goal. Our language has a small core, policy combinators are methods, policy analysis is based on push-button technology such as SAT solvers or BDDs, and propositional constraints express sets of access requests. Key to the simplicity of this approach is the ability to promote a constraint to a policy (those requests that the policy grants, respectively, denies), and to demote a policy to a constraint. The foundations of this work are rooted in Belnap's four-valued logic.

This is joint work with Glenn Bruns from Bell Labs, Naperville, Illinois.

Short Bio

Michael Huth received his degree of Diplom-Mathematiker from the Technical University of Darmstadt (Germany) in 1988. At Tulane University of Louisiana he then studied mathematics, computer science and philosophy of mind and was awarded a PhD in 1991. From 1991 to 1996 he conducted post-doctoral studies in semantics, theory of programming languages, and probabilistic model checking at Kansas State University, Imperial College London, the Technical University of Darmstadt, and the University of Birmingham (England). From 1996 to 2001 he was an Assistant Professor at the Department of Computing and Information Sciences at Kansas State University. In July 2001 he was appointed as Senior Lecturer in the Department of Computing at Imperial College London, where he now holds the post of Reader in Computer Science. His current research activities are in the areas: model checking and abstraction, quantitative methods for modeling and analyzing software artifacts, program analysis, access control, and digital identity management.

Implementing Prejudice in Trust Models: A Prototype

HS Venter[1], JHP Eloff[1], M Wojcik
Information & Computer Security Architectures Research Group
[1]Department of Computer Science, University of Pretoria, Lynnwood Road,
0002 Pretoria, South Africa
{hventer, eloff}@cs.up.ac.za

Abstract. Trust models are used to minimise the risk of sharing information between participants by evaluating the interactions between participants that are in a specific relationship. A trust value that represents the trustworthiness of this relationship is determined. Trustworthiness has a direct impact on the security requirements of relationships. The number of interactions required to determine a trust value, over and above the normal interactions, are in most cases too high and therefore hampering the successful implementation of trust models. In order to minimise the volume of interactions between participants, the work reported on in this paper employs the concept of prejudice filters. Prejudice is defined as a negative attitude towards communicating participants based on stereotypes. This paper discusses the practical implementation of prejudice filters as an extension to existing trust models. Empirical results demonstrate the advantage of employing prejudice filters in an example.

Keywords: Trust models, trust, prejudice filters, trust model prototype.

1 Introduction

Trust models have been proposed to minimise the risk of sharing and successfully analysing information [1, 2]. Such trust models rely on the abstract principle of trust for controlling which information is shared and with whom. Furthermore, trust models evaluate the participants of an interaction and assign a numerical value, known as a trust value, to the interaction. This trust value is used to determine the restrictions placed on the interaction and the nature of the information shared. It therefore has a direct relationship with the number and type of security mechanisms that are required for establishing a secure interaction. The process of trust analysis occurs in respect of all interactions that a participant running a trust model encounters and can lead to an overwhelming communication load on a network. Another potential problem that arises during the process of establishing trust involves the level of comprehensiveness required by the analysis process. Having a large number of strict rules defined in a trust relationship limits the communications a participant will be able to participate in, while at the same time adding to the analysis load. Furthermore, lower trust levels result in higher security requirements.

The authors of this paper are particularly interested in the confluence of trust and security. The primary aim is to investigate problems in the security domain where the solutions can benefit from the trust domain. This is dependant on investigating practical implementations of trust models which guided the authors to focus on the number of interactions required to establish trust levels. Therefore; the concept of prejudice filters was introduced in earlier work[3].

Prejudice is an extension of the concept of trust-building processes and is defined as a negative attitude towards an entity, based on stereotypes. It is important to note that the negative nature of prejudice allows a role for negative assumptions in evaluating trust.

Prejudice affects trust by allowing negative assumptions to be made about certain groups. These negative assumptions are based on prior knowledge and experience with such groups. All entities of a certain stereotyped group are placed in the same category. This not only allows assumptions to be made, but also simplifies the processing that is required before trust can be established [4]. A participant now needs to analyse only those attributes about which it does not have assumptions. A participant is also allowed to completely distrust another participant, simply because the latter falls into a category that is perceived as negative.

Many conceptual trust models, of which some already include the concept of prejudice filters, are available today. However, very few of these models have been successfully implemented and employed in software as found in industry today. It is therefore the objective of this paper to demonstrate the feasibility of a trust model implementation employing the concept of prejudice filters. The paper also discusses empirical results obtained from a prototype implementation.

The remainder of the paper is structured as follows: Section 2 focuses on related work, as well as on previous work done by the authors of this paper. Section 3, focuses on the design of a prototype to implement prejudice filters in trust models. Section 4 conveys the empirical results of the research and Section 5 concludes the paper.

2 Related Work

The authors refer to research work in three areas that are related to this paper: trust, trust models and prejudice filters.

2.1 Trust

Trust is a subjective concept – the perception of trust is unique to each individual. Even though there are several definitions of trust, there is general consensus about the need for trust. Trust is ultimately required to reduce complexity created by (for example) large complex environments. Trust allows for certain assumptions to be made and hence simplifies the information that an individual is required to analyse.

Nooteboom [5] defines trust as a four-place predicate, stating that (1) someone has trust, (2) in something, (3) in some respect (i.e. competence, intentions, benevolence) and (4) under some conditions. Gambetta [6] approaches trust in a more technical manner, formalising trust values as a range of values. Although this approach is rather ambiguous since the same value can be interpreted differently by different individuals, it provides scientists with a means by which to mathematically formalise what is inherently an abstract, intangible concept.

2.2 Trust and Security

The confluence of trust and security investigates how trust can play a role in searching for solutions in the security domain. The inherent properties of open distributed environments including amongst others web services, grid computing, cloud computing, and peer-to-peer networks, demand novel ways of implementing information security services such as access control. Confidence in open distributed environments is usually problematic and can be enhanced by security and trust. Wong and Sycara [25] showed that by adding security and trust to a multi agent system you increase user's confidence that the agents will behave in an expected and predetermined manner. They specifically address the security problems of agent masquerading and insecure communication. Another interesting area where trust and security are playing an increasingly important role is that of reputation systems. In [26] it is argued that the importance of ensuring the secure implementation of reputation-based systems cannot be over emphasized.

The authors of this paper are primarily interested in investigating the confluence of trust and security. However, the remainder of the paper at hand focuses on the practical implementation of trust models. Therefore the reader should note that security and trust would not be discussed in the remainder of this paper.

2.2 Trust models

Trust models are code constructs that rely on the concept of trust to assign a trust value to an interaction. This trust value is then used to control and restrict the interaction. Trust models have been proposed to allow participants who have not interacted previously to communicate with one another. These models further allow trust in participants to change as environments and circumstances change.

Different ways of establishing trust are critical to all trust models. These include, but are not limited to, recommendation, reputation, observation, institution and experience [7, 8, 9, 10, 11]. In order to change trust over time, trust models also update trust. A wide range of trust models have been proposed in literature. These include Aberer and Despotovic's P-Grid system for managing trust in a peer-2-peer environment [12]; the localised trust model for ad hoc networks by Luo et al. [13]; the computational model for trust and reputation proposed by Mui, Mohtashemi and Halberstadt [14]; and Abdul-Rahman and Hailes's trust reputation model [1]. Seeing that it is beyond the scope of this paper to elaborate on all of these models, the latter was focused on to demonstrate and compare empirical results regarding the implementation of prejudice filters.

The trust reputation model of Abdul-Rahman and Hailes [1] allows peers to share recommendations. Furthermore, this model is well structured and it implements the concepts of direct trust, recommendation and reputation. The model is cited by several experts who have made a worthy contribution towards research into trust [7, 12, 14, 15].

The model developed by Abdul-Rahman and Hailes is briefly summarised as follows: A subset of direct experiences and recommendations is summarised in order to obtain a trust value. Experiences are recorded in two separate sets with a view to differentiating between those that are a result of direct trust and those that are linked to a recommendation. These trust values are separated by the particular context for which they are formed, resulting in several trust values for a single participant, depending on the context. In order to obtain a direct trust value for a particular participant in a particular context, this model relies on a basic system of counters. For every participant within a specific context, a participant has four counters. The four counters used in this model are counters for very good, good, bad and very bad. They represent various trust levels, namely very trustworthy, trustworthy, untrustworthy and very untrustworthy. The counters are incremented as a result of the interactions in which a participant participates. The trust reputation model executes a MAX function on these four counters, thus returning the counter that has the highest value for that particular participant in a specific context. The trust counter with the highest value indicates the trust level to be assigned to that participant.

According to Ramchurn *et al*. [16] basic interactions among participants go through three main phases. These phases are negotiation, execution and outcome evaluation. Trust plays an essential part in all three of these phases. This is illustrated by Figure 1.

Two participants attempting to communicate with one another are first required to establish a communication link, usually initiated by one participant and accepted by another. This process initiates a negotiation process whereby two participants negotiate various parameters, such as the security level of information that is to be shared or the services for which permission will be granted, that will define boundaries of the interaction. A trust value for the interaction is defined through comprehensive analysis of logical rules. The simplest way of storing and implementing these rules is to have them present in a list that the participant accesses and processes. In Figure 1, storage of these rules occurs in the trust definition list.

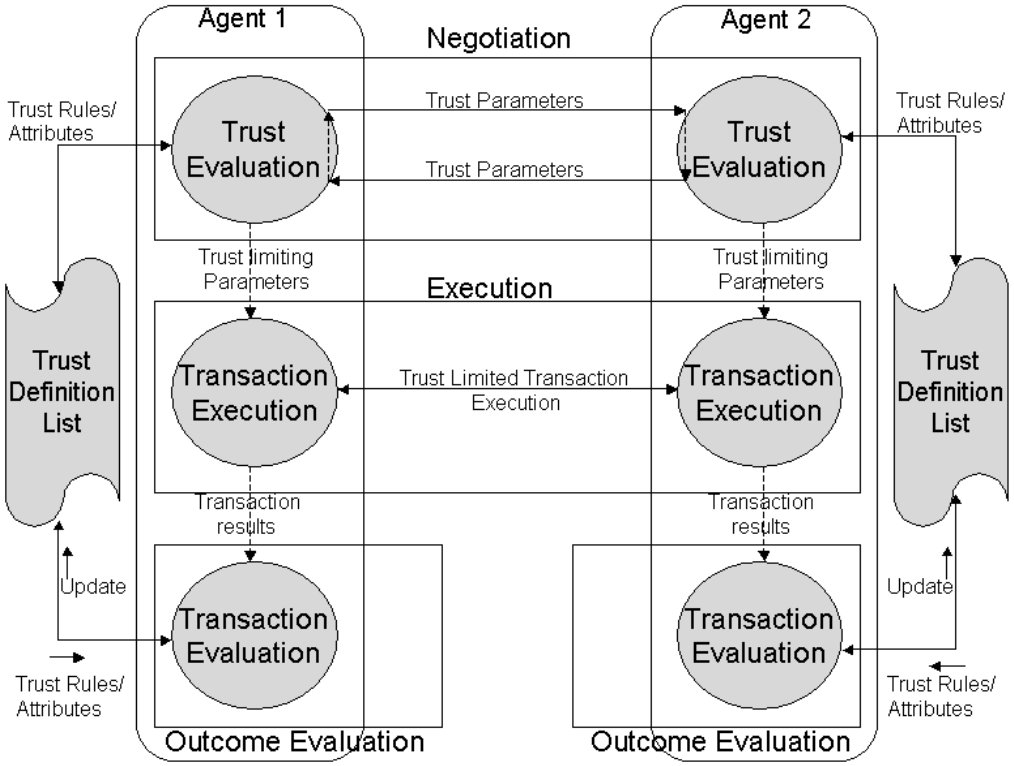


Fig. 1. *Operation of a participant using a trust model.*

The successful negotiation and establishment of a trust value triggers an analysis of the trust value. Provided the trust value is above a certain acceptable threshold, the transaction execution process is started. Trust models control the context of the interaction during the execution phase, limiting trust given and hence controlling which information or services are accessible and which are not.

Once transaction execution has terminated, the results of the interaction are sent to the transaction evaluation process. This process evaluates the results and updates the trust definition list in either a positive or negative manner. Negative updating of the logical rules occurs due to business transaction failure, while business transaction success will trigger a positive update.

The evaluation of trust among participants is a time-consuming process that requires comprehensive evaluation of the defined logical rules in order to attain an accurate trust value to be used during an interaction. Only once the trust value has been obtained, the participants will decide whether to participate in a transaction or not.

In a networking environment, the amount of possible participants that will request participation in such an interaction can be vast. To successfully assess another participant, participants pass several messages to obtain the required information that is to be analysed against the defined trust parameters. For instance, the formal model for trust in dynamic networks proposed by Carbone, Nielson and Sassone [17] passes delegation information between participants in order to create a global trust scheme. Delegation allows a particular participant to trust another participant, based on the fact that the other participant is trusted by participants that the participant in question trusts. This reliance on the passing of messages exposes the network to the possibility of network overload. Another potential problem arising during the process of establishing trust is the level of comprehensiveness required by the analysis process. Having a large number of strict rules define a trust relationship limits the communications a participant will be able to participate in, while at the same time adding to

the analysis load. Rules that are too generic open the system up to a higher level of risk by allowing a participant to participate in interactions with other participants that have not been fully analysed for trustworthiness.

Prejudice filters have been proposed to lessen the number of interactions that require comprehensive trust evaluation [18] so as to solve the problems mentioned above. Stereotyped grouping of interactions allows for characteristics to be assumed instead of evaluated in detail. It also allows trust evaluation to focus on characteristics that are not assumed, instead of evaluating the interaction against the entire list of logical rules that represent expectations.

2.3 Prejudice filters

In order to understand the concept of prejudice filters, an understanding of prejudice is required. Prejudice is an extension of the concept of trust-building processes and is defined as a negative attitude towards an entity, based on stereotype. It is important to note that the negative nature of prejudice allows negative assumptions in order to evaluate trust. Prejudice influences trust by allowing certain negative assumptions to be made about certain groups. These negative assumptions are based on prior knowledge and experience with such groups. All entities of a certain stereotyped group are placed in the same category, allowing assumptions to be made and simplifying the processing required before trust can be established [19]. This way a participant only needs to analyse attributes it does not have assumptions about in order to adjust trust value. A participant is allowed to completely distrust a participant simply because it falls into a category which it perceives as negative.

The concept of prejudice filters was introduced by Wojcik et al. [3]. In order to understand the remainder of this paper better, a brief summary of the concept of prejudice filters as introduced by Wojcik et al. [3] is provided in this section. Participants see prejudice filters as simplified trust rules that rely on the concept of prejudice to limit the number of interactions that a participant needs to analyse in detail. Prejudice filters rely on definitions of attributes that lead to distrusted interactions. For example, if a participant has interacted with another participant from a specific organisation and the interaction failed in terms of expectations, future requests from participants belonging to the same organisation will be discriminated against.

Figure 2 illustrates how prejudice filters extend a trust model, based on the work of Ramchurn et al. [16] as originally depicted in Figure 1. A prototype based on the trust model presented in Figure 2 was implemented to experiment with the concept of prejudice filters.

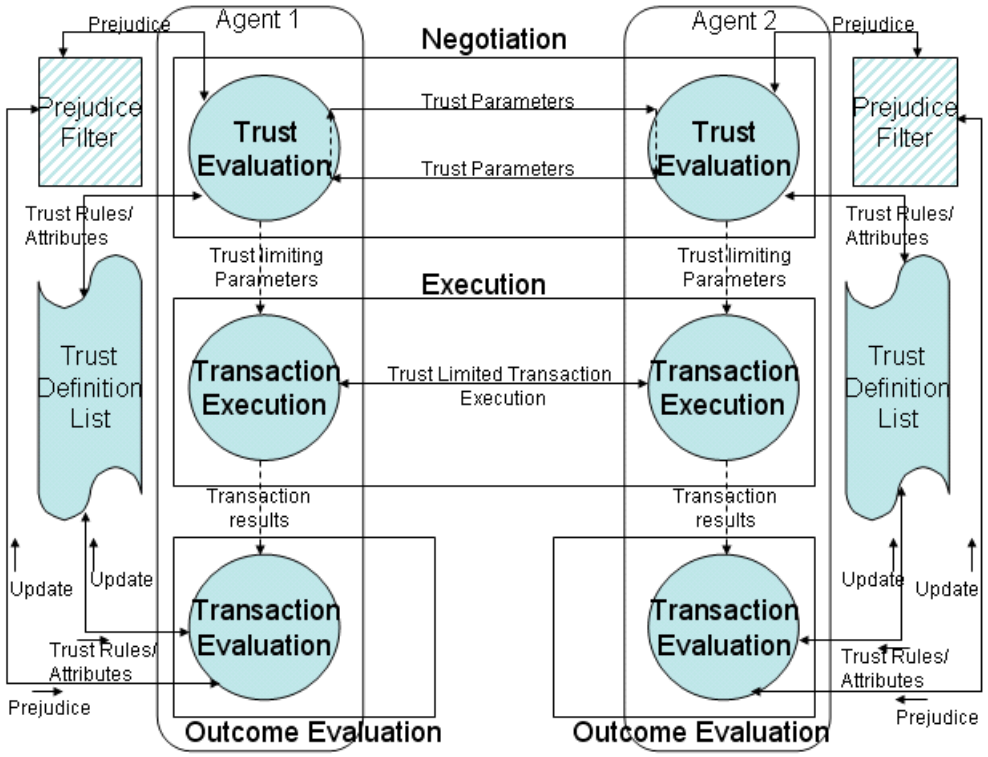


Fig. 2. *Operation of a participant using a trust model with prejudice filters [3].*

Existing trust models rely on various means of establishing trust, which include recommendation, reputation, third party reference, observation, propagation, collaboration, negotiation and experience [1], [2], [8], [10], [17], [20], [21], [22], [23], [24]. Based on these, five means of implementing prejudice filters have been identified by the authors in order to simplify the extension of existing models to include prejudice. These five are as follows [13]:

Learning: When using the learning filter, prejudice is not defined explicitly. A participant relies on 'first impressions' to learn prejudice. If an interaction fails, the participant analyses the interaction's attributes and looks for unique attributes of other interactions that were previously encountered and found to be successful. These unique attributes are used to create a category to be used as a prejudice filter.

Categorisation: A participant creates various categories that are trusted. If an interaction request does not fall into a trusted category, the participant filters out that interaction in a prejudiced manner. This can also be implemented in a reverse manner where a participant creates categories that are distrusted and filters out communications that fall into those categories. Categories can also be created to represent various levels of trust. Any interactions falling into such categories are assigned the default trust value associated with that particular category.

Policy: Policies define the operational environment in which a participant exists and affect parameters of interactions that are regarded acceptable. Policy-based prejudice filters out interactions with participants whose policies differ from the participant doing the filtering. One way of doing this is to request data on the country a participant resides in. Such data defines the laws and culture that bind business interactions for that participant, as well as controls the means in which data and confidentiality are is handled.

Path: Path-related prejudice allows a participant to refuse an interaction, simply because of the fact that the path of communication between two participants passes through a distrusted participant.

Recommendation: Participants that are trusted to make recommendations are known as recommender participants. Implementing prejudice by using recommendation allows a particular participant to only trust other participants that are trusted by the particular participant's recommender participants.

The above five filters can be incorporated into current trust models to extend their capability, while at the same time allowing for these filters to merge with a particular trust model's main philosophy. Just as some models use a combination of concepts to implement the concept of trust, interrelated filters can be implemented in different combinations in order to optimise their effectiveness [3].

3 Prototype Design

This section provides a functional description and explains the structural design of the prototype.

3.1 Functional description

As many trust models are or will be implemented on the Internet, specifically in a web services environment, we have chosen a virtual environment where one participant requests a service from another. The prototype implementation therefore emulates several participants in a given service-oriented environment. Since a participant is not allowed to perform its own service, it is required to request the service from another participant. If the other participant accepts this request, the service request is serviced.

It should be noted that, since the prototype runs as a simulation, the five filters, as mentioned in the previous section, are not all implemented. Instead, only one filter is implemented and merely simulated by means of timers. To do this, each participant in the prototype has a timer that records the length of time it takes to service a request. If participant A requests an interaction, it initiates a timer on its own side. It times the participant who is performing the request and stops the timer as soon as it receives a response. The time taken to receive the response defines the way in which participant A perceives the experience. The participant performing the service also has its own timer, because trust is considered to be unique to each participant participating in an interaction. The participant that performs a service initialises its timer when it sends a message that confirms the completion of the service and waits for acknowledgement. The time taken to receive such an acknowledgement determines the experience used to determine the trust levels.

According to Abdul-Rahman and Hailes's model [1], each participant possesses a set of trust counters pertaining to other participants that it knows and has interacted with. Since the trust counters refer to experiences that are very good, good, bad and very bad, they determine the level of trust a participant has in another participant. The counters are evaluated in the course of determining a trust level for another participant. The counter that returns the highest value when analysed by means of a MAX function determines the trust level given to an interaction. The four trust values associated with the counters are *vt* (very trustworthy), *t* (trustworthy), *u* (untrustworthy) and *vu* (very untrustworthy) and link to the four experience counters (vg, g, b and vb) respectively. In order to update the counters, each participant needs to evaluate the experience. Experience evaluation occurs by analysing the timers each participant possesses.

3.2 Structural design

The prototype is a console-based application written in C# and it runs on Microsoft.NET. Abdul-Rahman and Hailes's trust reputation model [1] was modified to include constructs that can be used by the prejudice filters. This was accomplished by storing the domain a

participant belongs to, as well as a participant's identity (ID) and context. A broad overview of the structure of the prototype is given in Figure 3.

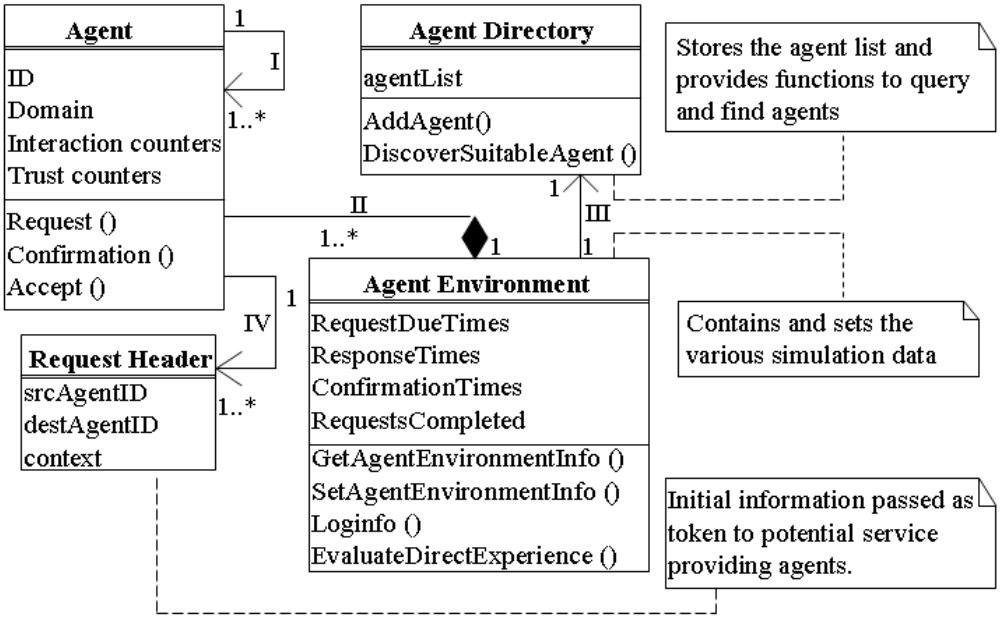


Fig. 3. *Class diagram of the prototype implementation*

The structure of the prototype consists of four key classes and the relationships between them. The four key classes are Agent, Agent Directory, Request Header and Agent Environment. The Domain value (in the Agent class) determines the environment in which a participant resides. Hence, this domain is used to implement the prejudice filter.

A participant also possesses two sets of counters. The first set of counters is labelled as interaction counters used to record statistical information. For example, these counters keep track of the total number of interactions that have been initiated and completed by a particular participant and other participants that it interacts with. The second set of counters (trust counters) keeps track of the participant's trust-related data. The set contains counters for vg (very good), g (good), b (bad) and vb (very bad), which record the experience grades of an interaction.

Participants communicate and interact with other participants, as illustrated by the relationship labelled 'I' in Figure 3. A single participant may interact with one or more other participants to accomplish a given task. The prejudice filters are implemented on the participant side and they are contained in some of the participant's methods. The Request() method allows a participant to request a service. Such a participant looks for a suitable participant by considering the trust counters it possesses for every other participant. If a suitable participant does not exist, the participant contacts the Agent Directory class and requests the Agent Directory class to return participants in the environment that it does not have trust counters for. Prejudice filters are implemented on the request side by allowing a participant to filter participants based on information it receives from the Agent Environment class. The Agent Environment class in turn receives this information from the Agent Directory class. The Accept() method allows a participant to accept service requests from other participants. Prejudice filters are also implemented on the accept side by allowing a participant to check the domain that a request is coming from. The Confirm() method allows participants to confirm receipt of a service and ultimately terminate a connection. Trust updates are performed on the request as well as the accept side, as both the participant that

requests a service and the participant that accepts a service are required to update the trust relationship according to their own perceptions.

The Agent Environment class is made up of one or more participants, and it controls the simulation data (labelled 'II' in Figure 3). The Agent Environment class controls the participants that have been defined, and it contains the various time-related data that the participants use. This time-related data includes RequestDueTimes (the exact time when a participant makes a request), ResponseTimes (the time it takes a participant to respond to a request) and ConfirmTimes (the time taken to confirm a service received).

The Agent Environment class uses the Agent Directory class (labelled 'III' in Figure 3) to keep track of all the participants within the environment. The Request Header class is simply a token that every participant possesses (labelled 'IV' in Figure 3) when that participant interacts with another participant.

4 Experimental Results

As an agent interacts with other agents, it gains experience and the experience gained influences the trust this agent has in other agents. In order to experiment with these experiences, the authors considered the impact of incremental trust establishment over time and the influence of experience. This was done by changing the number of iterations in the execution of the prototype. The simulation, presented in this paper, was run ten times *without* the prejudice filter and subsequently ten times *with* the prejudice filter, incrementing the number of iterations with each simulation. Table 1 records the results of the simulation without the prejudice filters and Table 2 records the results with the prejudice filters.

Table 1. Agent A's simulation results without prejudice filters

		Number of iterations									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Results	***vb***	3	3	4	4	2	3	3	3	14	6
	b	1	1	0	0	2	1	1	1	2	1
	g	0	0	0	1	0	0	0	0	0	1
	vg	3	5	7	6	9	8	9	11	1	14

Table 2. Agent A's simulation results with prejudice filters

		Number of iterations									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Results	***vb*(f)**	0	0	0	0	0	0	0	0	0	1
	***b*(f)**	0	0	0	0	0	0	0	0	0	1
	***g*(f)**	0	0	0	0	0	0	0	0	1	0
	***vg*(f)**	4	8	12	16	20	24	28	32	35	37

All the results recorded in Table 1 were from Agent A's perspective and illustrate the trust data recorded by Agent A. During each iteration, each agent made one request for a service. The rows of each table list the four counters that Agent A possesses. The value next to each counter represents the number of experiences that resulted in a particular experience grade for Agent A during the given simulation run. These counters record the sum of all the different experiences Agent A had with all other agents in the environment. Consequently we are interested in the total number of experiences, defined by a particular experience grade, of Agent A. For instance, the first data column in Table 1 shows that when the simulation was

run for a single iteration, Agent A had three vb (very bad), one b (bad), zero g (good) and three vg (very good) experiences overall.

The simulation with prejudice filters filtered out the agents that took a long time before responding. Table 2 demonstrates a clear decrease in bad experiences and an increase in good experiences overall. Figure 4 illustrates what happens to the number of interactions within each trust experience grade across the spectrum of *vb*, *b*, *g* and *vg*. The bold line represents the data recorded with the prejudice filter, while the dashed line represents the data recorded without the filter. Figure 4 clearly illustrates a drastic increase in the number of very good experiences when prejudice filters are used. As the number of iterations increased, the number of very good experiences increased more dramatically when using a prejudice filter than when not using a prejudice filter. The bad and very bad experiences decreased remarkably, because the domain that resulted in bad experiences due to long delays in ResponseDueTime and ConfirmationTime (as shown in Figure 4), was filtered out.

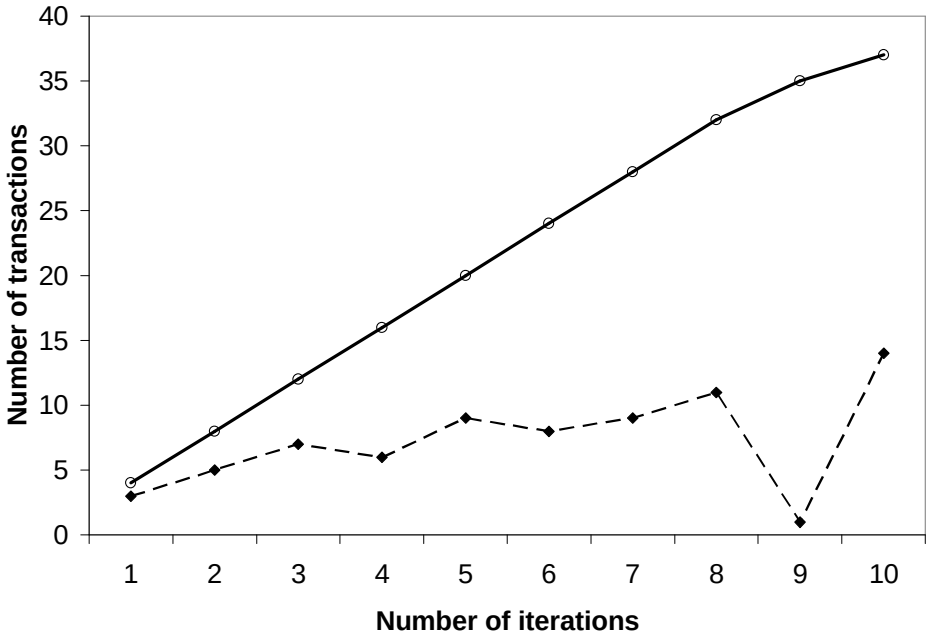


Fig. 4. *Comparing vg counters with and without prejudice filters*

5 Conclusion

In summary, one may perceive the notion of introducing prejudice filters for trust models as merely a variation from reputation-based trust models, namely negative reputation trust models. There are, however, substantial differences. The main advantage of implementing prejudice filters in trust models is the positive impact it has on the performance of a system. In contrast, reputation-based systems, be they positive or negative, still requires many more information sourcing activities and computational events resulting in high overheads.

The overall results indicated that prejudice filters have a positive impact on trust. Not only did the number of failed interactions drop remarkably, but the number of successful interactions also increased remarkably. The incorporation of prejudice filters in trust models helped to minimise communication loads in a network, as a significantly decreased amount of analysing needed to take place.

A meaningful extension of this research in future could involve the defining of more detailed and specific implementations of the various types of prejudice filters. This would include the standardisation of the protocols required to carry trust-related information, so that participants who run different trust model implementations can have a standard way of acquiring and identifying the data that is important to their specific implementations.

References

1. Abdul-Rahman, A., Hailes, S.: A distributed trust model. In: Proceedings of the 1997 Workshop on New Security Paradigms, Langdale, Cumbria, United Kingdom, pp. 48-60 (1998)
2. Ramchurn S.R., Sierra, C., Jennings, N.R., Godo, L.: A Computational Trust Model for Multi-Agent Interactions based on Confidence and Reputation. In: Proceedings of the 6th International Workshop of Deception, Fraud and Trust in Agent Societies, Melbourne, Australia, pp. 69-75 (2003)
3. Wojcik, M., Eloff, J.H.P., Venter, H.S.: Trust Model Architecture: Defining Prejudice by Learning. In: LNCS, vol. 4083, pp. 182-191. Springer, Heidelberg (2006)
4. Bagley, C., Verma, G., Mallick, K., Young, L.: Personality, self-esteem and prejudice. Saxon House, Teakfield Ltd, Westmead. Farnborough, Hants, England (1979)
5. Nooteboom, B.: Trust: forms, foundations, functions, failures, and figures. Cheltenham, Edward Elgar, p. 8 (2000)
6. Gambetta, D.: Can we trust trust? In: Gambetta, D. (ed), Trust, pp. 213-237. Oxford, Blackwell (1990)
7. Jøsang, A.: Prospectives for modelling trust in information security. In: Proceedings of the Australasian Conference on Information Security and Privacy, pp. 2-13 (1997)
8. Xiong, L., Liu, L.: A reputation-based trust model for peer-to-peer ecommerce communities. In: Proceedings of the IEEE Conference on Electronic Commerce, Newport Beach, CA, USA, pp. 275-284 (2003)
9. Esfandiari, B., Chandrasekharan, S.: On how agents make friends: Mechanisms for trust acquisition. In: 4th Workshop on Deception, Fraud and Trust in Agent Societies, Montreal, Canada, pp. 27-34 (2001)
10. Papadopoulou, P., Andreou, A., Kanellis, P., Martakos, D.: Trust and relationship building in electronic commerce. Internet Research: Electronic Networking Applications and Policy, vol. 11(4), pp. 322-332 (2001)
11. Abdul-Rahman, A., Hailes, S.: A distributed trust model. In: Proceedings of the 1997 Workshop on New Security Paradigms, Langdale, Cumbria, United Kingdom, pp. 48-60 (1997)
12. Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM), pp. 310-317 (2001)
13. Luo, H., Zerfos, P., Kong, J., Lu, S., Zhang, L.: Self-securing ad hoc wireless networks. In: Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC), pp. 567-574 (2002)
14. Mui, L., Mohtashemi, M., Halberstadt, A.: A computational model of trust and reputation. In: Proceedings of the 35th Hawaii International Conference on System Sciences, pp. 2431-2439 (2002)
15. Yu, B., Singh, M.P.: Distributed reputation management for electronic commerce. In: Computational Intelligence, vol. 18(4), pp. 535–549 (2002)
16. Ramchurn, S.R., Sierra, C., Jennings, N.R., Godo, L.: Devising a trust model for multi-agent interactions using confidence and reputation. In: Applied Artificial Intelligence, vol. 18, pp. 833-852 (2004)

17.Carbone, M., Nielsen, M., & Sassone, V.: A formal model for trust in dynamic networks. In: Software Engineering and Formal Methods. In: Proceedings of the First International Conference on 25-26 Sept., pp. 54-61 (2003)
18.Wojcik, M., Venter, H.S., Eloff, J.H.P., Olivier, M.S., Incorporating prejudice into trust models to reduce network overload. In: Proceedings of South African Telecommunications and Networking Application Conference (SATNAC 2005). SATNAC, Telkom, CD ROM Publication (2005)
19.Bagley, C., Verma, G., Mallick, K., Young, L., Personality, self-esteem and prejudice. Saxon House. , Teakfield Ltd, Westmead. Farnborough, Hants. England. ISBN: 0 566 00265 5 (1979)
20.Hultkrantz, O., Lumsden, K., E-commerce and consequences for the logistics industry. In: Proceedings for Seminar on "The Impact of E-Commerce on Transport." Paris (2001)
21.Patton, M.A., Josang, A., Technologies for trust in electronic commerce. In Electronic Commerce Research, vol 4, pp. 9-21 (2004)
22.Marx, M., Treur, J., Trust dynamics formalised in temporal logic. In: Proceedings of the Third International Conference on Cognitive Science, ICCS, pp. 359-362 (2001)
23.Jonker, C.M., Treur, J., Formal Analysis of Models for the Dynamics of Trust based on Experiences. In: Proceedings of MAAMAW'99, LNAI (1999)
24.Damiani, E., De Capitani di Vimercati, S., Samarati, P., Managing Multiple and Dependable Identities. IEEE Internet Computing, November-December (2003)
25. Chi Wong, H., Sycara, K., Adding Security and trust to multiagent systems. Applied Artifcial Intelligence, 14 :927È941, 2000
26. Roslan, I., Boyd, C., Josang, A., Russell, S., A Security Architecture for Reputation Systems. Lecture Notes in Computer Science, pp. 176-185 (2003)

Learning Trust in Dynamic Multiagent Environments using HMMs

Marie Elisabeth Gaup Moe, Mozhgan Tavakolifard and Svein J. Knapskog

Centre for Quantifiable Quality of Service in Communication Systems[1],
Norwegian University of Science and Technology,
marieeli,mozhgan,knapskog@Q2S.ntnu.no

Abstract

In open multiagent systems, agents are owned by a variety of stakeholders and can enter and leave the system at any time. Therefore, trust is a fundamental concern in effective interactions which is a key component of such systems. In this paper, we propose a trust model for autonomous agents in mulitagent environments based on hidden Markov models and reinforcement learning. By this combination, the reliability of the hidden Markov model will be improved since its parameters are re-estimated after training of the model with the reinforcement learning module.

1 Introduction

The rapidly changing environments of the Internet suffer from problems related to fragile trustworthiness of its millions of active entities, which can be humans or mobile agents. This problem is nontrivial, as more and more commercial transactions get carried out over the Internet. Therefore, devising an effective approach for verification of trustworthiness in such complex environments is essential, since trust mechanisms play a key role in the security of the entities. Also the trust establishment is nontrivial, since the traditional and social means of trust cannot be applied directly to the virtual settings of these environments, because in many cases the involved parties did not have any previous interaction. In such scenarios, trust management techniques may be used to stimulate service quality and acceptable user behavior in online markets and communities, and also sanction possible unacceptable user behavior.

Application of autonomous agents in large-scale open distributed systems presents a number of new challenges such as:

- Agents with different characteristics can enter the system and interact with one another.
- Each agent tries to maximize its individual utility because it represents a specific stakeholder with various objectives.
- Agents may change their identities on re-entering the system to avoid punishment for any past wrong doing.
- Agents should decide how, when, and with whom to interact without any guarantees that the interaction will actually achieve the desired benefits.

[1]"Centre for Quantifiable Quality of Service in Communication Systems Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT.
http://www.q2s.ntnu.no

Agents are faced with significant degrees of uncertainty in making decisions since it is impossible to obtain perfect information about the environment and the interaction partners properties. In such circumstances, agents have to establish appropriate trust in each other in order to minimize the impact of the uncertainty associated with interactions [10].

The goal of an agent in a dynamic environment is to make optimal trust decisions over time. Learning trust serves such a purpose by biasing the agent's action choices through information gathered over time. An agent can base its action choice on prediction of the other agents' behaviors or directly on the reward (the outcome of the interaction) received from them. Reinforcement learning is a systematic method that associates an agent's action with its rewards.

In reinforcement learning, an agent need not explicitly model other agents since its action can be directly based on the rewards. Thus this learning method is particularly useful for cases where agents have little knowledge about each other. An agent in a multiagent system may know little about others because information is distributed. Even when an agent has some prior information about others, the behavior of others may change over time. It is therefore natural to apply a learning algorithm.

Our model consists of trust estimation and trust learning modules. The former and latter are constructed from *Hidden Markov Models* (HMM) and *Reinforcement Learning (RL)*, respectively. The model parameters of the HMM are re-estimated after having learnt about its environment from the reinforcement learning module. The proposed method enables us to improve the model reliability when dealing with a dynamic environment that changes over time.

2 Related Work

In [5] a trust model using a Markov model is proposed. In this work the Markov chain is defined as the chain of aggregated reputation values corresponding to a sequence of consecutive time slots. The current state vector shows the repute value of the reputation queried at time slot N.The Markov matrix of a given agent denotes the probability of that agent transiting from one trustworthiness level to another trustworthiness level based on its past behavior captured using the Markov chain. In order to determine the probability of an agent transiting from trustworthiness level A to trustworthiness level B, based on the Markov chain, they use the ratio of the number of times that agent has transited from trustworthiness level A to trustworthiness level B to that of the total number of time that the agent has transited from trustworthiness level A to any other trustworthiness level. The future state vector of the agent is determined by multiplying the current state vector with the Markov matrix. The same authors also proposed a method for determining the effectiveness of their Markov model for predicting the future trustworthiness value of a given agent by utilizing simulation methods [6]. This paper presents the simulation method that they employed in order to determine the effectiveness of the Markov model in detail.

In [12], the authors compare the effectiveness of probabilistic computational trust systems. They conclude that most probabilistic trust models are unrealistic, as the models allows for no dynamic behavior, and outline the idea of a trust model based on a hidden Markov model to cope with this problem. In [13], the authors have developed a hidden Markov model based approach to measuring an agent's reputation as a recommender. This approach models chained recommendation events as an HMM. The features of the trust model are: (1) no explicit requirement of chained recommendation reputations; (2) flexible recommendation network with presence of loops; and (3) integration of learning speed into trust evaluation reliability. In [7] an architecture for trust management in ubiquitous environments that deals with digital signatures and user presence in a uniform framework is proposed. This architecture includes inferences about user presence from incomplete sensor signals based on an HMM.

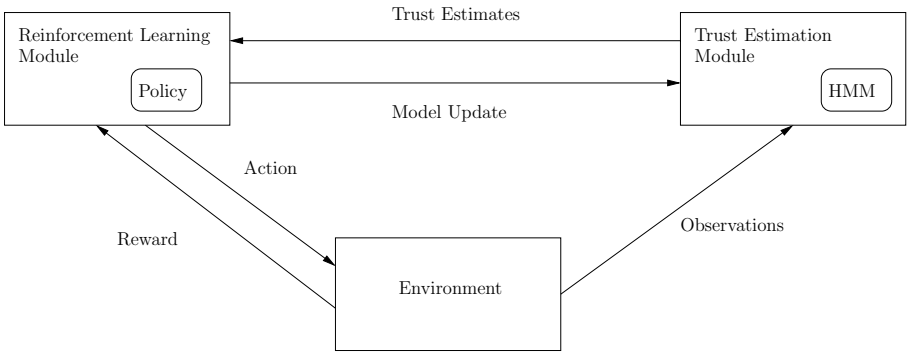


Figure 1: Architecture of the proposed trust model

3 The Proposed Model

Our model consists of trust estimation and trust learning modules constructed from *Hidden Markov Models* (HMM) and *Reinforcement Learning (RL)* as depicted in Figure 1. The model parameters of the HMM are re-estimated after having learnt about its environment from the reinforcement learning module. The proposed method enables us to improve the model reliability when dealing with a dynamic environment that changes over time. Similar approaches combining reinforcement learning and HMMs applied to motion recognition can be found in [4] and [2].

In the following sections we will start with describing some assumptions and limitations of this work, thereafter we present the stochastic modeling approach and details of the hidden Markov modeling.

3.1 Model States

An agent can be in a *trusted*, *neutral* or *untrusted* state at any given time. An agent is in an untrusted state if it has been behaving in a malicious way in previous interactions, it is in a trusted state if it has shown good behavior. If its behavior has been a mixture of good and bad, or if it has not yet given any signs of its behavior, it is in the neutral state.

When an agent joins the system it is most likely in a neutral state, but there is also a small probability that it is in the untrusted state. Since it has not yet interacted with any of the other agents, no positive observations of its behavior have been made, so it can not be in a trusted state.

After joining the system the agent has a probability of becoming trusted or untrusted as it is interacting with other agents, depending on its behavior. We model trust as a dynamic variable, changing with time. This allow us to capture the behavioral characteristics of agents that are behaving good for a certain time, but then suddenly start misbehaving.

3.2 Modeling of Agent Trustworthiness

In this section we will present an HMM for the trust relationship of agents in a multiagent system.

We model the agent interaction as a stochastic process. This means that we assume that there is a random time interval between each agent interaction and that the behavior of a node is only dependent on the current state of the node. The *state* of an agent can be characterized by whether or not it is behaving in a malicious manner in its interactions with other agents.

When using a continuous time Markov model to model the state of an agent, we make the following assumptions; all information about the agent is contained in the state, observations are independent given the current state, and state occupation time is negatively exponentially distributed.

When agents are interacting, an agent makes its opinion about the trustworthiness of the other agent based on the outcome of the interaction. After a random time interval these two agents meet again, and based on their belief about the other agent's trustworthiness they may decide whether or not to make an interaction. Since an agent's behavior can be changing with time it is not necessarily the case that an agent is in the same state as it were at the last encounter. An agent can only do its best guessing about the trustworthiness state of an other agent based on its own previous direct experiences with said agent and *recommendations* from other agents in the system. This means that the system state is hidden, and hence we use the HMM approach.

The system we consider is a multiagent system and we want to use the model to estimate the behavior of each single agent. An agent in the system rates all of the other agents after an interaction and uses an HMM per agent to decide and predict whether or not another agent is malicious. The HMM is updated from observations, that is the ratings after direct experiences or recommendations requested from other agents.

An HMM consists of a finite set of N hidden states $S = \{s_1, \ldots, s_N\}$ with an associated probability distribution. The state of the monitored agent is described by a discrete time Markov chain $\boldsymbol{x}_k = x_1, x_2, \ldots$ where $x_k \in S$ is the possibly hidden state of the node at sampling instant k. $\boldsymbol{P}_k = \{p_{ij}^k\}$ is the set of state transition probabilities, $p_{ij}^k = P(x_{k+1} = s_j \mid x_k = s_i)$, $1 \leq i, j \leq N$, where x_k is the current state of the system. $\pi = \{\pi_i\}$ is the initial state distribution, where $\pi_i = P(x_1 = s_i)$, $1 \leq i \leq N$. The output from the agent ratings is classified by the set of observation symbols $V = \{v_1, \ldots, v_M\}$. Let $\boldsymbol{y}_k = y_1, y_2 \ldots$ denote the sequence of observations, where $y_k \in V$ is the observation made at sampling instant k. The HMM consists of two stochastic processes; the hidden process $\boldsymbol{x}_k$, and the observable process $\boldsymbol{y}_k$ that depends on $\boldsymbol{x}_k$. The relation between $\boldsymbol{x}_k$ and $\boldsymbol{y}_k$ is described by the probability distribution matrix $\boldsymbol{B} = \{b_j(m)\}$, where $b_j(m) = P(y_k = v_m \mid x_k = s_j)$, for $1 \leq j \leq N$, $1 \leq m \leq M$. See for instance [8] for a more extensive introduction to HMMs.

In our model we define three states, that can be characterized by the behavior of the agent, thus $N = 3$ and each individual state is denoted $S = \{s_1, s_2, s_3\}$. The first state is the *trusted* state s_1, where the agent is not showing any malicious behavior, the second state is the *neutral* state s_2, where the outcome of interactions can be ambiguous, the third state is the *untrusted* state s_3 where the agent is showing malicious behavior.

We have not made any assumptions about time between observations, and there is no direct relation between observations and state-changes. As a consequence the system could have made zero, one or more transitions during the time between to successive observations.

The time when observation number k is produced is denoted t_k. Time between observation $k-1$ and observation k is denoted $\delta_k = t_k - t_{k-1}$.

3.3 State Probability Distribution

The transition rate matrix $\Lambda = (\lambda_{ij})$ is describing the dynamics of the system. To simplify the notation of equations and algorithms we will use i and j instead of s_i and s_j. The relation between system states and the transition rates is given by

$$\lambda_{ij} = \begin{cases} \lim_{dt \to 0} \dfrac{P(\boldsymbol{x}(t+dt) = j | \boldsymbol{x}(t) = i)}{dt} & \text{if } i \neq j \\ \sum_{j \neq i, j=1}^{N} -\lambda_{ij} & \text{if } i = j \end{cases}. \tag{1}$$

Since observations are received at irregular intervals, the running transition probabilities $p_{ij}^k = P(x(t+\delta_k) = j | x(t) = i)$ depend on the time since last observation δ_k, and have to be calculated each time an observation is received. The running transition probability matrix $\boldsymbol{P}_k = (p_{ij}^k)$ can be derived from Kolmogorov's equations [11] as follows

$$\boldsymbol{P}_k = e^{\Lambda \delta_k}. \tag{2}$$

For large state spaces this calculation can be quite expensive, but in our case the state space is small, and the calculations inexpensive. Let $\gamma_k = (\gamma_k(i))$ denote the state probability distribution at time t_k given all observations received until time t_k, $\gamma_k(i) = P(x_k = i | \boldsymbol{y}_k)$ where $\boldsymbol{y}_k = y_1, \ldots, y_k$. We will use ten observation symbols $V = \{v_1, v_2, \ldots, v_{10}\}$, where the first five symbols are the ratings an agent make after a direct interaction and the last five symbols are ratings received as recommendations from other agents. The ratings are given in the form of trustworthiness values ranging from 1 to 5, where 1 corresponds to "very trustworthy", 2 to "trustworthy", 3 to "moderate", 4 to "untrustworthy", and 5 to "very untrustworthy". Algorithm 1 is used to update the current state distribution γ_{k-1}, based on the following inputs: k the observation index, $\gamma = \gamma_{k-1}$ the current state distribution, $y = y_k$ the current observation, and $\delta = \delta_k$ the time between the current observation and last observation. In addition to the dynamic variables listed above, the following parameters are assumed to be available for the algorithm: the transition rates Λ, the initial state distribution π , and the two observation probability matrices $\boldsymbol{B}^\psi$, where $\boldsymbol{B}^1$ is used for the direct observations $v_1, \ldots, v_5$, and $\boldsymbol{B}^2$ is used for the implicit observations in the form of recommendations $v_6, \ldots, v_{10}$.

Algorithm 1 Update state probability distribution

Require: $k, \psi, \gamma, y, \delta$
  $P_k \leftarrow e^{\Lambda\delta}$
  $B \leftarrow B^\psi$
  if $k = 1$ **then**
    for $i = 1$ to N **do**
      $\alpha(i) \leftarrow b_i(y)\pi_i$
      $\gamma(i) \leftarrow \frac{b_i(y)\pi_i}{\sum_{j=1}^N b_j(y)\pi_j}$
    end for
  else
    for $i = 1$ to N **do**
      $\alpha(i) \leftarrow b_i(y) \sum_{j=1}^N \gamma(j) p_{ji}^k$
    end for
    for $i = 1$ to N **do**
      $\gamma(i) \leftarrow \frac{\alpha(i)}{\sum_{j=1}^N \alpha(j)}$
    end for
  end if
  return γ

Algorithm 1 was originally proposed in [3], it is based on dynamic programming and uses a set of temporary variables. During the processing of observation y_k the value stored in $\alpha(i)$ represents the following probability $\alpha(i) = P(\boldsymbol{y}_k, x_k = s_i)$, also known as the *forward variable*. By using dynamic programming in the estimation of γ, the complexity of an update is reduced from $O(2kN^k)$ for a straight forward calculation, to $O(N^2)$. Scaling of the $\alpha(i)$ is used in order to prevent problems related to underflow, for more details see the forward-backward procedure described in [8]. It should be noted that Algorithm 1 is an on-line algorithm and very efficient,

it does not require the agents to keep any history of past observations in memory. The history of observations and the values of some of the running variables are, however, required for the more computationally expensive re-estimation of model parameters as explained later.

4 Learning of Model Parameters

The parameters of an HMM are usually set by offline training of the model with a large data set. Since we want the model to reflect the more realistic dynamic behavior of the multiagent system and also optimize the agents' trust-related behavior, we will use an online learning of the HMM parameters with *reinforcement learning*. Reinforcement learning (RL) [14] is a machine learning technique for solving decision problems of mapping actions to states based on interactions with the environment. The actions of an agent in the multiagent system could for instance be whether or not it should interact with another agent, based on its belief about the state of the other agent derived from the HMM. Such a mapping from state to action is called a *policy*. In RL the agents will learn policies based on feedback from the environment that is calculated based on a *reward function*.

A simple reward function for the multiagent trust model can be defined as follows:

1. If an interaction was made, and the agent's rating of the other agent's behavior was given the values 1, 2 or 3, a positive reward is given.
2. If an interaction was made, and the agent's rating of the other agent's behavior was given the values 4 or 5, a negative reward is given.
3. If no interaction was made, a zero reward is given.

The RL framework also includes a *value function* $Q(s, a)$ which estimates the reward obtained if action a is performed in state s.

Q-learning [15] is a well-known RL algorithm that updates the value function in each step so that the agent policy converges to the optimal one. Q-learning works even though the state transition probabilities are unknown to the agent. In our approach we will use the output of the Q-learning to improve the HMM by updating the state transition rate matrix according to the learned optimal policy.

Since the current state of an agent is hidden in our trust model, we will instead use the state probability distribution γ that is calculated after each observation, and learn the function $Q(\gamma, a)$. A variant of the Q-learning algorithm suitable for this case where the current state is only partially observable, and accounting for the fact that the domain of Q is not discrete and finite, can be found in [1]. Following this approach we associate a value $q(i, a)$ with each hidden state s_i, and approximate $Q(\gamma, a)$ as

$$Q(\gamma, a) \approx \sum_{i=1}^{N} \gamma(i) q(i, a).$$

Learning Q is done by adjusting all the q values after each action a and immediate reward r according to the Q-learning rule:

$$q(i, a) = (1 - \beta\gamma_k(i))q(i, a) + \eta\gamma_k(i)(r + \sigma \max_a Q(\gamma_{k+1}, a)), \tag{3}$$

where η is the learning rate and σ is a discount factor. The learning proceeds as follows, when an agent encounters another agent it will get the current state probability distribution γ_k belonging to this particular agent from its corresponding HMM and execute the action with

the largest $Q(\gamma_k, a)$. After the action is performed, the agent receives the reward, the next step state probability distribution γ_{k+1} is output from the HMM, and the Q-learning updating rule from Equation 3 is applied. The process is repeated at the next encounter between the agents. It should be noted that the observations to the HMM coming from recommendations will not result in actions or rewards, only the direct experiences in the form of agent interaction will trigger the reinforcement learning module in the trust model.

When the agent encounters another agent for the first time, the parameters of the HMM will be set to default values, and the model might not properly predict the system dynamics. A newcomer to the system should not be expected to be in the trusted state, as such a starting point would encourage agents to change their identities and re-enter the system frequently. It is therefore better to assume that agents are most likely neutral or untrustworthy to start with, and then they have to prove themselves trustworthy by behaving good in the interactions.

As the agent learns about the behavior of the other agent through direct experience and recommendations, the HMM parameters should be updated in order to improve the predictiveness of the model. We suggest that the state transition rates Λ and the observation probabilities $\boldsymbol{B}$ of the HMM are updated after a predetermined number of Q-learning steps, e.g. by the Baum-Welch algorithm which finds the maximum likelihood parameter estimate. See [8] for a detailed explanation of the Baum-Welch algorithm.

Given a sequence of K observations the Baum-Welch algorithm makes use of the *backward variable* $\beta_k(i) = P(y_{k+1}, \ldots, y_K | x_k = s_i)$, which is the probability of the observation sequence from next step and until the end given that we are in the state s_i at time-step k. The backward variable is found inductively by setting $\beta_K(i) = 1$ and then recursively calculating $\beta_k(i) = \sum_{j=1}^{N} p_{ij} b_j(y_{k+1}) \beta_{k+1}(j)$. The re-estimation procedure calculates the joint probability $\xi(i,j) = P(x_k = s_i, x_{k+1} = s_j | \boldsymbol{y})$ of being in state s_i at time-step k and state s_j in time-step $k+1$ as

$$\xi(i,j) = \frac{\alpha_k(i) p_{ij} b_j(y_{k+1}) \beta_{k+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_k(i) p_{ij} b_j(y_{k+1} \beta_{k+1}(j))}.$$

For re-estimation of p_{ij} according to Baum-Welch we use

$$p_{ij} = \frac{\sum_{k=1}^{K-1} \xi(i,j)}{\sum_{k=1}^{K-1} \gamma_k(i)},$$

where the nominator is the expected number of transitions from state s_i to state s_j, and the denominator is the expected number of transitions from state s_i. For re-estimation of the observation symbol probabilities the following equation is used

$$b_j(m) = \frac{\sum_{k=1, \text{s.t.} y_k = v_m}^{K} \gamma_k(j)}{\sum_{k=1}^{K} \gamma_k(j)},$$

where the nominator is the expected number of times in state s_j and observing the symbol v_m and the denominator is the expected number of times in state s_j.

Algorithm 2 implements the re-estimation of the parameters. In order to avoid problems related to underflow we use scaling of the backward variable as described in [8] and [9]. The proof of correctness of the scaling procedure is given in the Appendix A. Algorithm 2 takes as input all the K different α_k and γ_k vectors, which are calculated by Algorithm 1, as well as all the different $\boldsymbol{P}_k$. This means that these values need to be stored in the agent's memory together with the history of observations for the re-estimation procedure. To simplify the implementation we will combine the two observation probability matrices into one single matrix $\boldsymbol{B}$ for the re-estimation algorithm. The observation probabilities are modeling the uncertainties

Algorithm 2 Re-estimation of parameters

Require: $\boldsymbol{P}, \boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{B}, K$
  for $i = 1$ to N **do**
    $\hat{\beta}_K(i) \leftarrow \frac{1}{\sum_{i=1}^{N} \alpha_K(i)}$
  end for
  for $k = K - 1$ to 1 **do**
    for $i = 1$ to N **do**
      $\hat{\beta}_k(i) \leftarrow \frac{1}{\sum_{j=1}^{N} \alpha_k(j)} \sum_{j=1}^{N} p_{ij}^k b_j(y_{k+1}) \hat{\beta}_{k+1}(j)$
      for $j = 1$ to N **do**
        $\xi^k(i,j) \leftarrow \gamma_k(i) p_{ij}^k b_j(y_{k+1}) \hat{\beta}_{k+1}(j)$
      end for
      $\hat{\gamma}_k(i) \leftarrow \sum_{j=1}^{N} \xi^k(i,j)$
    end for
  end for
  for $i = 1$ to N **do**
    for $j = 1$ to N **do**
      $p_{ij} \leftarrow \frac{\sum_{k=1}^{K-1} \xi^k(i,j)}{\sum_{k=1}^{K-1} \hat{\gamma}_k(i)}$
    end for
  end for
  for $m = 1$ to M **do**
    for $j = 1$ to N **do**
      $b_j(m) \leftarrow \frac{\sum_{k=1,\text{s.t.} y_k = v_m}^{K} \hat{\gamma}_k(j)}{\sum_{k=1}^{K} \hat{\gamma}_k(j)}$
    end for
  end for
  return $\boldsymbol{B}, \boldsymbol{P}$

associated with observations. This means that when we re-estimate $b_j(m)$ elements associated with the direct observations, i. e. the agent's own ratings after interactions, we are evaluating the reliability of the agent itself when it comes to making good trust decisions. The elements associated with the recommendations are re-estimated as an evaluation of the reliability of the recommending agents. In this context we do not model the trustworthiness of each recommender separately, but this could be done as an extension of the model.

5 Conclusions and Future Work

We have proposed a novel trust model for multiagent systems where the goal of an agent is to make optimal trust decisions over time in a dynamic environment. An agent bases its action choice on a prediction of the other agents' behaviors according to the HMM trust estimation module following the Q-learning greedy policy. Since this is a policy that tends to the optimal one over time, the model learning algorithm will be training the HMM with sequences of observations that will positively impact the end goal.

As this is just a preliminary theoretical model, without any simulation results yet, there are many directions of research that may be explored to improve our work. We would like to see if it is possible to additionally train the model to making the best decisions about when to ask other agents for recommendations. Application of the model to a variety of trust scenarios and a comparison to other proposed trust models is of course the prime interest of our future work.

References

[1] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI Press, 1992.

[2] K. Hamamoto, K. Morooka, and H. Nagahashi. Motion Recognition By Combining HMM and Reinforcement Learning. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2004.

[3] Kjetil Haslum, Marie Elisabeth Gaup Moe, and Svein Knapskog. Real-time Intrusion Prevention and Security Analysis of Networks using HMMs. In *Proceedings of the 4th IEEE LCN Workshop on Network Security (WNS)*. IEEE, 2008.

[4] M. A. T. Ho, Y. Yamada, and Y. Umetani. An HMM-based Temporal Difference Learning with Model-Updating Capability for Visual Tracking of Human Communicational Behaviors. In *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition (FGR'02)*. IEEE, 2002.

[5] F. K. Hussain, E. Chang, and T. S. Dillon. Markov model for modeling and managing dynamic trust. In *Proceedings of the 3rd IEEE International Conference on Industrial Informatics, INDIN'05*, pages 725–733. IEEE, 2005.

[6] F. K. Hussain, E. Chang, and T. S. Dillon. Quantification of the Effectiveness of the Markov Model for Trustworthiness Prediction. In *Proceedings of the International Conference 9th Fuzzy Days*. Springer, 2006.

[7] J. Noda, M. Takahashi, I. Hosomi, H. Mouri, Y. Takata, and H. Seki. Integrating presence inference into trust management for ubiquitous systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 59–68. ACM Press New York, 2006.

[8] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[9] Ali Rahimi. An Erratum for "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". http://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html, 2000.

[10] S.D. Ramchurn, D. Huynh, and N.R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(01):1–25, 2005.

[11] Sheldon M. Ross. *Introduction to Probability Models*, chapter Continuous-Time Markov Chains, pages 349–390. Academic Press, New York, 8th edition, 2003.

[12] V. Sassone, K. Krukow, and M. Nielsen. Towards a Formal Framework for Computational Trust. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects (FMCO 2006)*, volume LNCS 4709, page 175. Springer, 2006.

[13] W. Song, V.V. Phoha, and X. Xu. The HMM-Based Model for Evaluating Recommender's Reputation. In *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, pages 209–215. IEEE, 2004.

[14] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.

[15] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Appendix A Scaling of the forward and backward variables

We want to use scaling of the forward variable and the backward variable, to avoid problems related to underflow. Since, without scaling, these variables consist of a large number of terms of value significantly less than 1. This appendix will give a detailed explanation of the scaling procedure for the implementation of the Algorithms 1 and 2, as proposed in [8] and [9]. We have that

$$\alpha_k(i) = \begin{cases} \pi_i b_i(y_k) & \text{if } k = 1 \\ b_i(y_k) \sum_{j=1}^{N} \alpha_{k-1}(j) p_{ji} & \text{if } k > 1 \end{cases} \tag{4}$$

where $\alpha_k(i)$ is the forward variable without any scaling. We want to compute the scaled forward variable

$$\hat{\alpha}_k(i) = C_k \alpha_k(i), \tag{5}$$

with scaling coefficient

$$C_k = \frac{1}{\sum_{j=1}^{N} \alpha_k(j)}. \tag{6}$$

Let us use the notation $\bar{\alpha}_k(i)$ for the running value of the forward variable as it is used in the implementation before scaling, and c_k for the running value of the scaling coefficient. The recursion for calculating the scaled forward variable can then be expressed as:

Initialization:

$$\bar{\alpha}_1(i) = \alpha_1(i)$$

$$\hat{\alpha}_1(i) = \frac{\alpha_1(i)}{\sum_{j=1}^{N} \alpha_1(j)}$$

For $k > 1$:

$$\bar{\alpha}_k(i) = b_i(y_k) \sum_{j=1}^{N} \hat{\alpha}_{k-1}(j) p_{ji}$$

$$c_k = \frac{1}{\sum_{j=1}^{N} \bar{\alpha}_k(j)}$$

$$\hat{\alpha}_k(i) = c_k \bar{\alpha}_k(i)$$

We want to prove that this recursion realizes the scaling as expressed in Equation 5. For $k = 1$ the scaling coefficient is $c_1 = C_1 = \frac{1}{\sum_{j=1}^{N} \alpha_1(j)}$, so the scaling is exactly as in Equation 5. For $k > 1$ we can use a proof of induction as follows:

$$\begin{aligned} \bar{\alpha}_k(i) &= b_i(y_k) \sum_{j=1}^{N} \hat{\alpha}_{k-1}(j) p_{ji} \\ &= b_i(y_k) \sum_{j=1}^{N} C_{k-1} \alpha_{k-1}(j) p_{ji} && \text{(by application of Equation 5)} \\ &= C_{k-1} \alpha_k(i) && \text{(by application of Equation 4)} \end{aligned}$$

This gives us the relation

$$c_k = \frac{1}{\sum_{j=1}^{N} \bar{\alpha}_k(j)} = \frac{1}{C_{k-1} \sum_{j=1}^{N} \alpha_k(j)} \tag{7}$$

We can then express $\hat{\alpha}_k(i)$ as

$$\hat{\alpha}_k(i) = c_k \bar{\alpha}_k(i) = \frac{C_{k-1}\alpha_k(i)}{C_{k-1} \sum_{j=1}^{N} \alpha_k(j)} = \frac{\alpha_k(i)}{\sum_{j=1}^{N} \alpha_k(j)}$$

so the recursion used is indeed realizing the scaling as given in Equation 5, which was what we wanted to show. Furthermore, by combining Equations 6 and 7, we get the relation

$$C_k = C_{k-1} c_k = \prod_{\kappa=1}^{k} c_\kappa$$

Recall the definition of the backward variable:

$$\beta_k(i) = \begin{cases} 1 & \text{if } k = K \\ b_i(y_{k+1}) \sum_{j=1}^{N} \beta_{k+1}(j) p_{ij} & \text{if } k > K \end{cases}$$

For the scaling of the backward variable let us define the scaling coefficient

$$D_k = \prod_{\kappa=k}^{K} c_\kappa$$

which gives us the relation

$$C_k D_{k+1} = \prod_{\kappa=1}^{k} c_\kappa \prod_{\kappa=k+1}^{K} c_\kappa = C_K$$

Let us denote by $\hat{\beta}_k(i)$, the scaled backward variable

$$\hat{\beta}_k(i) = D_k \beta_k(i) \tag{8}$$

and let us denote by $\bar{\beta}_k(i)$, the running backward variable as it is used in the implementation before scaling. The recursion for calculating the scaled backward variable is given by

Initialization:

$$\bar{\beta}_K(i) = \beta_K(i) = 1$$

$$\hat{\beta}_K(i) = D_K = c_K$$

for $k < K$:

$$\bar{\beta}_k(i) = b_i(y_{k+1}) \sum_{j=1}^{N} \hat{\beta}_{k+1}(j) p_{ij}$$

$$\hat{\beta}_k(i) = c_k \bar{\beta}_k(i)$$

Note that the running value of the scaling coefficient will be the same c_k as was used in the scaling of the forward variable. For $k = K$ the Equation 8 is trivially fulfilled. The correctness

of the recursion for $k < K$ can be shown by induction as follows

$$\begin{aligned}\bar{\beta}_k(i) &= b_i(y_{k+1}) \sum_{j=1}^{N} \hat{\beta}_{k+1}(j) p_{ij} \\ &= b_i(y_{k+1}) \sum_{j=1}^{N} D_{k+1} \beta_{k+1}(j) p_{ij} \\ &= D_{k+1} \beta_k(i)\end{aligned}$$

we can then rewrite $\hat{\beta}_k(i)$ as

$$\hat{\beta}_k(i) = c_k \bar{\beta}_k(i) = c_k D_{k+1} \beta_k(i) = D_k \beta_k(i)$$

hence, the recursion is realizing the scaling in Equation 8.

Protecting VoIP Services Against DoS Using Overload Control

Dorgham Sisalem
Tekelec
Berlin, Germany
dorgham.sisalem@tekelec.com

John Floroiu
Tekelec
Berlin, Germany
john.floroiu@tekelec.com

September 26, 2008

Abstract

Like any other Internet-based service VoIP servers can be the target of denial of service attacks. While operators will certainly deploy various mechanisms to prevent such attacks, detecting and preventing DoS attacks is not a trivial task. Attacks can be disguised as legitimate VoIP traffic so distinguishing between a denial of service attack or a sudden surge in traffic due to some event is not always possible. Hence, VoIP servers need to incorporate mechanisms that would deal with the attack in a manner that would not lead to a complete service interruption. In this paper, we propose a new approach called the Receiver-based SIP Overload Control Algorithm (R-SOCA) which aims at stabilizing the behavior of a VoIP server during overload situations and preventing a complete collapse of the service. R-SOCA is designed so that it takes the cause of the overload as well the nature of the overloaded resource into consideration. R-SOCA was implemented in a SIP proxy and tested under various conditions. The test results show the ability of this algorithm to stabilize the performance of SIP proxies even under a denial of service attack.

1 Introduction and Motivation

VoIP servers constitute the core components of any VoIP infrastructure. These servers are responsible for processing and routing VoIP signaling traffic and supporting various advanced services such as call forwarding or voicemail. From a technical point of view, a Voice over IP service resembles to a greater extent an email service than a traditional telecommunication service. VoIP components use the same physical infrastructure as email services, the same transport protocols and very often the same operating systems and hardware. Hence, it is only natural to expect that a VoIP system can suffer from the same security threats as an email system or any other system in the Internet.

The goal of a denial of service (DoS) attack is to prevent legitimate users from using the attacked VoIP servers and to prevent them from initiating or receiving calls. This can be achieved by flooding a VoIP server with a high number of useless requests that deplete one or more resources of the host such as bandwidth, memory or CPU. Thereby, the VoIP server would be overloaded and would not have sufficient resources left for serving legitimate requests. Work done in [9] suggests that overload situations not only reduce the performance of a VoIP server but can finally lead to a complete standstill of the complete VoIP service. To withstand sudden increases in traffic or temporary lack of resources, VoIP servers need to implement overload control mechanisms that try to prevent a complete depletion of their resources.

Currently, most VoIP services are based on the session initiation protocol (SIP), [12]. SIP, however, does not offer sufficient mechanisms for handling overload situations. Hence, in this paper, we describe a new overload control scheme designed to be used by SIP-based VoIP

servers. The scheme called Receiver-based SIP Overload Control Algorithm (R-SOCA) aims at reducing the load on VoIP servers by shedding traffic proportionally to the measured overload status. R-SOCA is designed so that it takes the cause of the overload as well the nature of the overloaded resource into consideration. R-SOCA is designed to improve the performance of the SIP proxies without the need to standardize new features of SIP or to realize some cooperation or feedback between SIP proxies.

In Sec. 2 we take a brief look at possible causes of overload and the current status of congestion control for SIP. R-SOCA is then presented in Sec. 3. To test the performance of R-SOCA, R-SOCA was implemented as part of a SIP proxy. The test results are described in Sec. 4. In our performance investigation we investigate the capability of R-SOCA to stabilize the behavior of a SIP proxy under various attack models. Sec. 5 finally summarizes the paper and describes the open issues that will need to be solved in future work. Note that in the context of this paper only SIP proxies will be mentioned. However, the proposed solution applies equally well to all other kinds of SIP components such as PSTN gateways or back-2-back user agents.

2 Background and Related Work

In its simplest form a SIP-based VoIP service consists of user agents (UA), proxies and registrar servers. The UA can be the VoIP application used by the user, e.g., the VoIP phone or software application, a VoIP gateway which enables VoIP users to communicate with users in the public switched network (PSTN) or an application server, e.g., multi-party conferencing server or a voicemail server.

The registrar server maintains a location database that binds the users' VoIP addresses to their current IP addresses.

The proxy provides the routing logic of the VoIP service. When a proxy receives SIP requests from user agents or other proxies it also conducts service specific logic, such as checking the user's profile and whether the user is allowed to use the requested services. The proxy then either forwards the request to another proxy or to another user agent or rejects the request by sending a negative reply.

A SIP proxy acts in either statefull or stateless mode. In the statefull mode, the proxy forwards incoming requests to their destination and keeps state information about each forwarded request until either a reply is received for this request or a timer expires. If the proxy did not receive a reply after some time, it will resend the request. In the stateless mode, the proxy would forward the request without maintaining any state information. In this case the user agents would be responsible for retransmitting the request if no replies were received. As the statefull behavior reduces the load on the user agents and provides the service provider with greater session control possibilities, e.g., forwarding the request to another destination if the first one did not reply, statefull SIP proxies are usually used by VoIP providers.

With regard to the SIP messages we distinguish between requests and replies. A request indicate the user's wish to start a session (INVITE request) or terminate a session (BYE request). We further distinguish between session initiating requests and in-dialog requests. The INVITE request used to establish a session between two users is a session initiating request. The BYE sent for terminating this session would be an in-dialog request. Replies can either be final or provisional. Final replies can indicate that a request was successfully received and processed by the destination. Alternatively, a final reply can indicate that the request could not be processed by the destination or by some proxy in between or that the session could not be established for some reason. Provisional responses indicate that the session establishment is in progress, e.g, the destination phone is ringing but the user did not pickup the phone yet.

SIP proxies constitute the core components of a VoIP service and will have to handle signaling traffic arriving from thousands if not millions of user agents. In this paper we will therefore concentrate on providing a solution for handling congestion scenarios that might influence the behavior of SIP proxies.

From the brief description of SIP-based VoIP services, it is obvious that there are two main resources that might get overloaded at a SIP proxy, namely, CPU and memory. CPU resources are used for parsing incoming messages and executing service specific tasks which might include writing logging information, reading from a database or evaluating a user's profile for example. When acting in transaction statefull mode, memory will be consumed at the SIP proxy for maintaining various transaction state information related to the request. This memory will be dedicated for this transaction until either a final reply was received for this request or some timer expires. [12] specifies that a proxy is supposed to keep these state information for several seconds or even a few minutes depending on the exchanged requests and replies. Besides these two main resources several other resources are essential to the proper working of a SIP proxy. This includes the number of free processes or threads in a multi-process implementation, see [15], the number of busy ports and disk space. Due to space limitations we will not consider these resources in this paper.

CPU and memory could get overloaded at a SIP proxy due to various reasons such as:

- Denial of service (DoS) attack: DoS attacks on a SIP proxy can take different forms and target either the memory consumption of the server or the CPU -or both [6].
  - Flooding attacks: With these kinds of attacks, an attacker generates a large number of SIP requests. Even if these requests end up being dropped by the proxy, the proxy will first have to parse and process them before deciding to either forward, reject or drop them. Depending on the number of generated requests, such attacks can misuse a large portion oft the CPU available to the proxy and reduce the amount of CPU available for processing valid requests.
  - Memory attacks: This is a more intelligent kind of attack in which the attacker sends valid SIP requests that are forwarded by the proxy to destinations that do not answer properly. With each forwarded request the proxy will maintain some transaction state. If the destination of these requests does not answer at all, then the proxy will keep on trying for some time, 32 seconds, the so called Timer B in [12], before it can delete the state information. If the destination answers with a provisional response but not with a final one, then the proxy will have to keep the transaction state for at least 3 minutes, the so called Timer C in [12].
- Flash crowds: Flash crowd scenarios describe a sudden increase in the number of phone calls in the network. An example for this is when thousands of users want to vote on a TV show. This sudden increase in the number of calls will result in an increase in the required CPU and memory at the server.
- Unintended traffic: Software errors or configuration mistakes can cause one or more user agents or SIP proxies to send unintentionally multiples of the amount of the traffic they usually generate. Even though the excess traffic is not generated with malicious intent it is just as useless as DoS traffic and can just as well cause an overload situation.
- Software errors: Software errors include memory leak problems or infinite loops. Memory leak would deplete the available memory in a similar manner as a memory DoS attack. An infinite loop could block the proxy from serving SIP requests.

The session initiation protocol (SIP) is specified in the RFC 3261 [12]. While this specification describes the behavior of a SIP proxy in general, it does not provide much guidance on how to react to overload conditions. [12] indicates that a server that is not capable of serving new requests, e.g., because it is overloaded, could reject incoming messages by sending a 503 (service unavailable) reply back to the sender of the request. Additionally, the 503 reply includes a retry-after header which indicates to the sender of the request when it can try to resend the request to this server. This would signal the sender to try forwarding the rejected request to another proxy and not to use the overloaded proxy for the retry-after time. In case the overloaded server is contacted by a lot of users then using different values of retry-after would ensure that the traffic arriving at the proxy would only increase gradually. However, in a lot of deployment scenarios a proxy is contacted only by a few other proxies or gateways. In this case using the 503 approach would result in a stop and go traffic behavior at the overloaded proxy which would lead to an oscillative and instable over all network behavior. Also directing the traffic to other proxies would usually cause these proxies to become overloaded themselves. Hence, using 503 is more likely to relief the overloaded proxy only when it is facing a lot of user agents and in cases of internal errors, e.g., hardware or software problem, that would prevent the proxy from performing in the expected manner.

In [11] the author discusses the drawbacks of using the 503 reply for indicating overload and the requirements an overload control scheme should fulfill. Among these requirements, is that an overload control scheme should enable a server to support a meaningful throughput under all circumstances, should prevent forwarding traffic to other servers that might be overloaded themselves and should work even if not all servers in the network support it.

Hilt describes in [5] an approach for handling overload based on having the overloaded proxy exchanging load information with its neighboring proxies. The neighboring proxies would then adjust the amount of traffic they send to the overloaded proxy based on this information. While this approach seems to be promising, no simulative or measured results are reported for this work. Further, this approach requires the definition of new headers and replies, making it usable mainly between proxies that implement this solution.

[2] describes an approach for avoiding overload by separating different types of SIP messages into different queues and allocating specific shares of the CPU to the different queues. This approach can help in preventing overload situations when for example the overload is caused by a certain type of messages. However, this approach does not help against memory attacks or flooding attacks that use different kinds of SIP requests. In a related manner, in [10] the author also discusses the possibility of using different queues to prioritize certain messages. The authors in [8] presented initial results comparing a SIP network without overload control, with the built-in SIP overload control and with a rate-based overload control scheme. However, their paper does not discuss issues of denial of service or resource specific overload handling mechanisms.

Using simulations, the authors in [14] compare between a number of window and rate based overload control mechanisms based on the feedback approach. While the results are very promising the applicability of the presented solutions for TCP is not discussed. Also these schemes require cooperating proxies and are not optimized to deal with denial of service attacks or to protect the server's memory. Similar to some of the approaches discussed above R-SOCA is also designed to prioritize certain traffic in case of overload. However, in addition, R-SOCA takes into consideration the causes of the overload, e.g., DoS attack or flash crowd, in its reaction to overload. R-SOCA further does not require cooperating neighbors so it could be introduced without requiring other servers to support overload mechanisms. Finally, R-SOCA can be applied to servers using TCP as well as UDP.

3 Receiver-based SIP Overload Control Algorithm (R-SOCA)

As discussed in Sec. 2, a SIP proxy is said to be overloaded if one or more of its resources is having a value above some maximal limits. Going above these limits can destabilize the system and even lead to complete failure, see [7]. Hence, the goal of any overload control scheme is to reduce the load on the resources.

Besides aiming at reducing the load on the overloaded resources, R-SOCA was designed with the following additional goals in mind:

1. Keep the amount of used resources at the SIP proxy at predictable levels.
2. While the stability of the SIP proxy is not endangered, aim at serving the engineered load. Engineered load (E) is defined as the number of requests the system should be able to deal with. This includes the number of session initiating INVITE messages (E_I), number of in-dialog requests (E_D), e.g., the requests that are exchanged as part of a dialog such as BYE and re-INVITE messages, number of REGISER messages (E_R) and the number of out-of dialog messages such as INFO, OPTIONS or MESSAGE for example (E_O). The values for the engineered traffic are defined based on the experience of the manufacturer of the SIP proxy and the expected usage scenario.
3. Prioritize running sessions. This is based on the assumption that a user would be more annoyed if his call gets interrupted or can't be terminated and he has to pay more because his BYE request was rejected by the SIP proxy than if he could not make a call in the first place.
4. Identify traffic that might be part of a DoS attack and penalize it with a higher rate than other traffic
5. A SIP proxy implementing R-SOCA will interoperate seamlessly with other SIP entities that have no knowledge of R-SOCA.

3.1 Probabilistic Message Rejection

There are two main resources that need to be protected from getting overloaded: memory and CPU. R-SOCA defines for each resources (R), e.g., memory and CPU, two thresholds; a minor (R_{min}) and a major threshold (R_{max}). Once the minor threshold is exceeded, R-SOCA starts gradually reducing the load on the SIP proxy by rejecting incoming requests with the rejection probability of P_T^R. Once the maximum threshold is exceeded, all incoming requests are rejected. The minimum threshold (R_{min}) is set to the amount of resources required to serve the so called engineered traffic (E). The engineered traffic, is the amount of calls a SIP proxy is expected to receive and process successfully under normal operational conditions, e.g., no DoS attacks or flash crowd scenarios. The maximum threshold (R_{max}) is set to the amount of resources required to handle some worst case scenarios such as flash crowds.

The major characteristics of R-SOCA can be described as follows:

- At fixed time intervals T the current load status R_T for each resource R is measured and its rejection probability (P_T^R) is determined.

$$P_T^R = \begin{cases} 0 & R_T \leq R_{min} \\ \frac{R_T - R_{min}}{R_{max} - R_{min}} & R_{min} < R_T < R_{max} \\ 1 & R_T \geq R_{max} \end{cases} \quad (1)$$

- The rejection probability to be used by the SIP proxy (P_T) is then determined as the maximum value of all P_T^R. To avoid rapid changes in the rejection probability, an averaged rejection probability ($\widehat{P}_T$) is determined as a wighted moving average over time with

$$\widehat{P}_T = \widehat{P}_{T-1} \times (1 - \sigma) + P_T \times \sigma \tag{2}$$

  For smoothening the rejection probability a σ of 0.6 was used. This value proved after a couple of measurements as the best choice between rapid reaction to sudden changes in the traffic and smooth changes of the rejection value.

  As the rejection probabilities are determined as moving averages, they will not converge to 0 or 1. Hence, $\widehat{P}_t$ is used as follows:

  - If $\widehat{P}_t \leq 0.05$, then the system in not considered overloaded and no incoming requests will be rejected
  - With $0.05 < \widehat{P}_t$, then the system is considered overloaded
  - If any P_T^R had a value of 1 then the system in considered severely overloaded and $\widehat{P}_T$ is set to 1.

- when the system is in overload status INVITE requests and out of dialog requests such as REGISTER and MESSAGE will be rejected with probability of $\widehat{P}_T$. To ensure that emergency calls are processed even under overload situations, emergency requests should not be rejected. In-dialog requests are not rejected.

- when the system is considered severely overloaded, e.g. $\widehat{P}_T$ set to 1, then all incoming requests will be rejected

The rejection is achieved by sending a 500 reply including a retry-after header indicating that the user agent should resend the request after T_{retry} seconds. The 500 replies are sent in stateless manner. That is, no state information is maintained for them and the negative reply is not retransmitted if no ACK was received. In case the 500 reply is lost the sender would resend the INVITE request. In case the sender receives the 500 reply it will send an ACK request. As the proxy will not find an appropriate transaction state for the ACK it can just drop it.

3.2 Denial of Service Protection

Now, it is naturally possible that a DoS attack would consist of sending large amounts of requests that look like in-dialog requests such as BYE messages for example. In such a case, the above mentioned steps would not lead to a reduction of the overload. To accommodate this scenario, for each type of in-dialog requests, the number of such requests (D) is counted and compared to the expected number of requests of this type (D_{en}). Depending on the used traffic model, see [13], each type of requests will constitute a certain share of the overall requests. So for example if 1000 calls per second are being processed successfully, e.g., the INVITE messages are getting a positive reply from the destination, then we can expect to receive on the average also 1000 BYE requests per second. If more than the expected level of a certain type of requests was received then this can be seen as an indication of a DoS attack and the requests arriving in excess of the expected traffic should be dropped. Hence, the approach of Sec.3.1 is extended as follows:

- During each measurement interval T, the number of arriving data packets (D) for each type of in-dialog requests is counted.

- Each type of requests that exceeds its engineered value will be dropped with a probability of $((D - D_{en})/D)$. Re-INVITE messages will be rejected instead of dropped with a 500 reply including a retry-after header indicating that the user agent should retry the re-invite after T_{retry} seconds.

Thereby, the rejection rate of the requests which are suspected to be part of a DoS attack will be set proportionally to the number of requests sent in excess of the engineered load. This rejection probability would, however, still allow the forwarding of the engineered load.

For the case of in-dialog messages, rejecting requests can often have negative side effects. In IMS [1] the session establishment requires the exchange of an INVITE plus a number of PRACK and UPDATE requests. In case the INVITE message was accepted but the PRACK or UPDATE requests were rejected then the session establishment would be prolonged by at least the value of the retry-after indicated in the 500 reply. Also, a user agent that has sent a BYE might not even wait for the duration of the retry-after time (T_{retry}) indicated in the 500 replies.

As dropped messages will be normally retransmitted, the retransmitted messages will have some chance of passing an overloaded server. That is, with the exponential retransmission approach used by SIP, see [12], a user agent would retransmit an in-dialog request up to 9 times before quitting the session.

The re-INVITE messages can be rejected, as the user agents have to maintain the transaction state anyway and delaying will in general not have a considerable negative effects on the user's session.

The probability that all retransmitted requests sent by a user agent get dropped can be determined as p^n with p being the drop or rejection probability at the proxy and n bing the number of the retransmitted request. Hence, to ensure that the probability that at least one of the retransmitted requests pass through the overloaded server with a probability of p_{pass}, p should be set in the following manner:

$$p^n < p_{pass} \tag{3}$$

$$p < \sqrt[n]{p_{pass}} \tag{4}$$

with n set to 10.

D_{en} is set to different values depending on the type of the request. For requests that are sent only once in a dialog such as BYE or UPDATE, D_{en} is set to the following value:

$$D_{en} = max(E_I, I) \times B_{en} \tag{5}$$

with I as the average value of successful session initiating INVITE requests and B_{en} as a burstiness factor used to accommodate sudden bursts of traffic.

Re-invite messages occur more often during a session. In this case, D_{en} would be set to

$$D_{en} = max(E_I, I) \times B_{en} \times N_r \tag{6}$$

with N_r as the expected number of re-invite messages during a session and can be estimated as the average duration of a session divided by the session update timer, see [3]. The session update timer indicates the frequency with which reINVITE messages are sent.

3.3 Burstiness Protection For CPU

On one hand, Internet traffic is rather bursty in nature, see [4]. On the other, resources such as CPU should only be measured as average values over some time intervals. Further, continuously

calculating the rejection rate and updating the system behavior would require a considerable amount of resources. Hence, it is possible that in between two measurement points, a large burst of signaling packets arrives that would require more processing resources than are available to the system. To allow for bursts of traffic an internal traffic counter is used that counts the number of received requests. Once the number of received requests of one type of engineered traffic, e.g., E_I, E_D, E_R, E_O exceeds a certain threshold, B_{en} times the engineered traffic for that kind of requests with ($B_{en} \geq 1$) the excess traffic will be rejected. That is, if for example if during the measurement interval T more than ($E_I \times B_{en}$) INVITE requests were received the proxy will start rejecting all incoming session initiating INVITE requests.

The CPU value is measured every T seconds. As the CPU values C_T oscillate, we use an averaged value of the CPU $\widehat{C}_T$ and is determined as

$$\widehat{C}_T = \widehat{C}_{T-1} \times (1 - \sigma) + C_T \times \sigma \tag{7}$$

For our measurements we used a σ value of 0.6.

4 Performance Evaluation

In this section we test the performance of R-SOCA under different traffic conditions.

4.1 Test Environment

To test the performance of R-SOCA we extended a SIP proxy[1] with the required logic. The testbed itself consists of a number of senders initiating calls to a number of receivers through. The SIP requests are sent from the senders to two forwarding SIP proxies which then forward the traffic to the target SIP proxy. The SIP proxies are connected to the network over a Gb/s link, see Fig. 1. The senders and receivers are emulated using the SIPP tool[2]. To be able to test R-SOCA the target SIP proxy was extended with the following features:

- Measure the CPU load each T seconds
- Continuously measure the memory used by the proxy
- Count the amount of each type of incoming requests (D)
- Execute the R-SOCA logic, e.g., determine the rejection probability ($\widehat{P}_T$) and the drop and rejection probability of in-dialog messages
- Probabilistically reject or drop incoming requests if required so by R-SOCA. When there is the need to decide if a request is to be forwarded or rejected a random number between 0 and 1 is generated. If this number is lower than the rejection probability then the request is rejected.

The proxy was configured to support an engineered load (E_I) of 2000 calls per second. The burstiness factor (B_{en}) was set to 1.7.

4.2 CPU Performance Evaluation of R-SOCA

The minimum threshold (C_{min}) was set to 20% of the available CPU. This was determined by measuring the CPU needed for handling the engineered traffic. The maximum threshold (C_{max}) was set to 40%.

[1]The SIP Express Router was used. See www.iptel.org for more information

[2]SIPP is an open source traffic generator available under http://sipp.sourceforge.net/

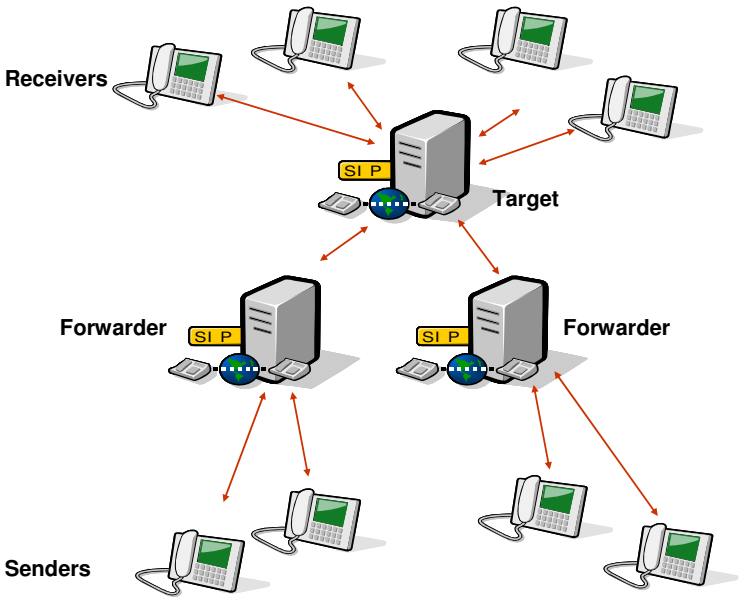


Figure 1: Test environment

Performance of R-SOCA under Heavy Load In this scenario the performance of R-SOCA is tested when subjected to an incoming call rate of around 10000 session initiating INVITE requests per second which is much higher than the supported engineered load (E_I=1000). Actually the amount of received traffic results exceeds both the minimum and maximum thresholds. As seen in Fig. 2 R-SOCA manages to keep the CPU load around the maximum threshold (C_{max}). From Fig. 3 we can observe that the CPU load is distributed between processing and rejection of the incoming calls. On the average around 1500 calls per second are still successfully processed and maximally 1700 which is the maximum accepted burstiness level.

Performance of R-SOCA under BYE DoS Attacks In the previous tests DoS attacks were initiated by sending a large number of INVITEs. In the scenario tested here, the emulated DoS attack consists of a sender sending a constant load of 12000 BYE messages per second. With the DoS protection mechanism of R-SOCA, BYE requests that arrive in excess of the expected values are dropped. With an engineered traffic of 1000 calls per second and the burstiness factor set to 1.7, the expected number of BYEs would be 1700. In general a dropped BYE would be retransmitted by the user agent. However, as it is simpler for an attacker to generate multiple BYE requests instead of keeping a state machine that would be needed for conducting the SIP retransmission behavior, it is assumed here that an attacker will not retransmit a dropped BYE.

Further, p_{pass} was set to 10% which results in a maximal allowed drop rate of 79.5%. Thereby, only 10% of the sent BYE transactions will not be able to terminate successfully.

Fig. 4 depicts the behavior of the proxy when it is receiving 12000 BYE messages and 1000 INVITE requests per second. The DoS protection mechanism will result in dropping BYE requests both from the attacker as well as from legitimate users. The dropped BYE requests which were sent by legitimate users are retransmitted leading to an overall rate of incoming BYE requests of around 18000 BYEs per second. From Fig. 5 we can observe that even under

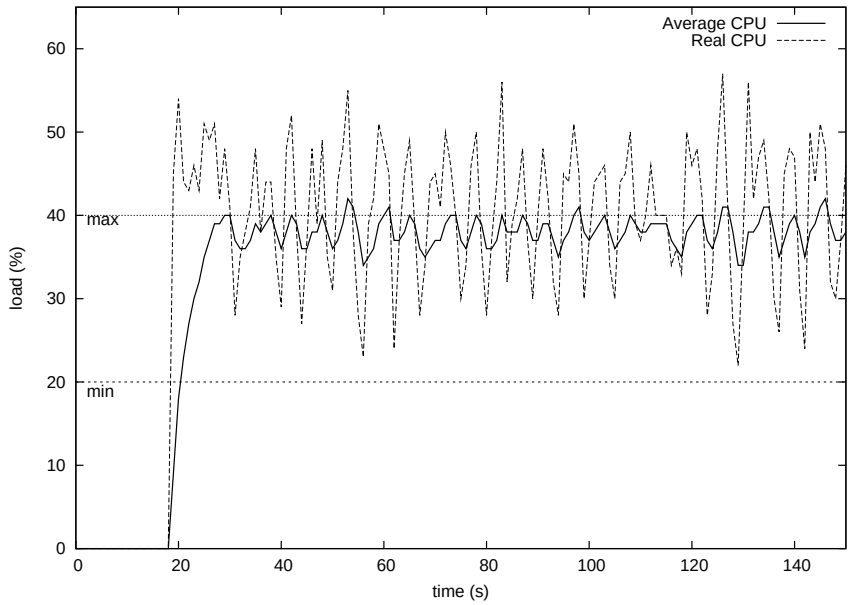


Figure 2: CPU consumption of R-SOCA under heavy load

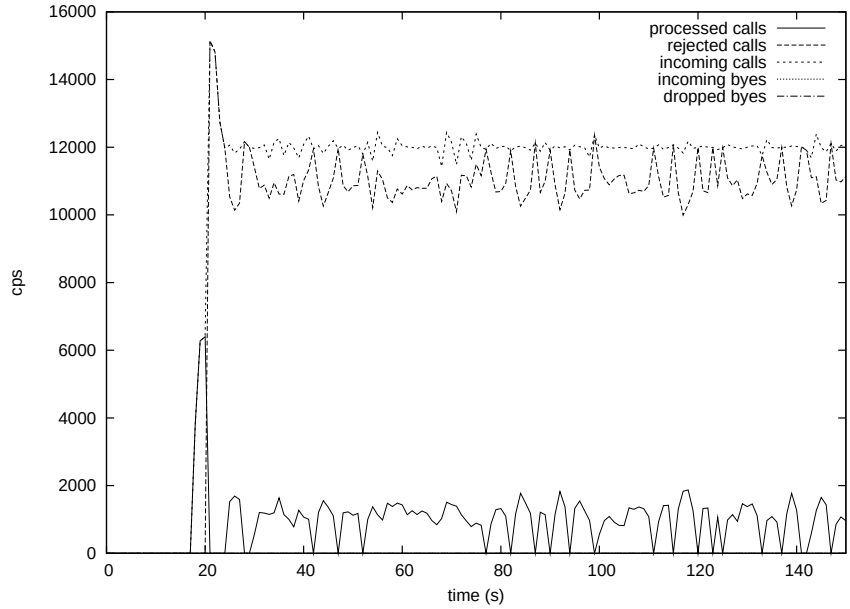


Figure 3: Call processing behavior of R-SOCA under heavy load

Sent Invites	Successful calls	Successful termination
196911	161193	150150

Table 1: Overall performance of R-SOCA under BYE DoS attack

this heavy load of incoming messages, the overall CPU rate is at around 23% which is only minimally higher than the used C_{min}.

Tab. 1 summarizes the results of the measurement. Out of nearly 200000 initiated calls around 15% of the calls were rejected. Out of the successful calls 94% were successfully terminated. 6% of the calls could not be terminated successfully as the sent BYE requests and their retransmissions were dropped by the overloaded proxy. Note that by increasing the value of the maximally expected BYE requests or using a smaller p_{pass} values would reduce the number of unsuccessful terminations. However, this would mean that a larger number of malicious BYE requests are processed by the proxy and hence would lead to a higher CPU usage which would then incresase the call rejection rate.

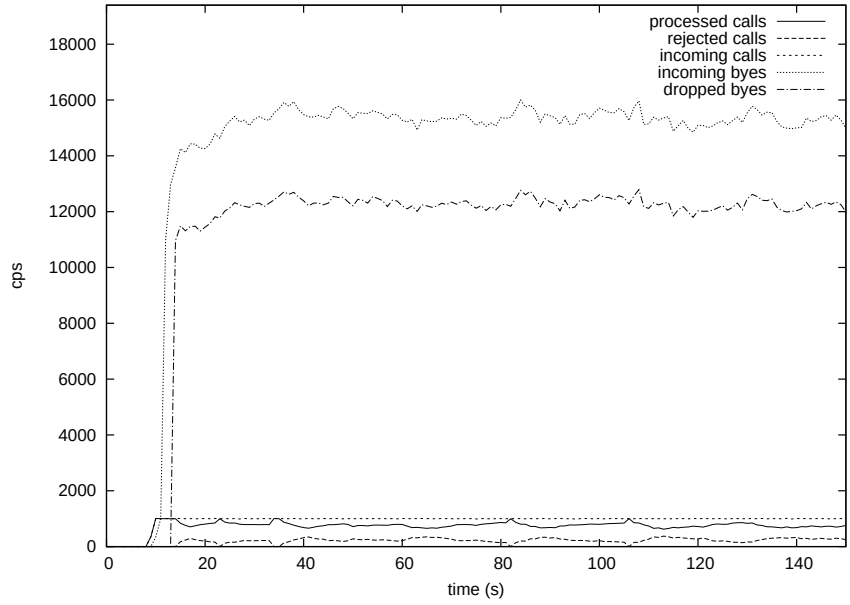


Figure 4: Performance of R-SOCA under BYE DoS attack

Fig. 6 depicts the behavior of the proxy when it is receiving 12000 BYE messages and 2000 INVITE requests per second, e.g. double the engineered traffic. We can observe that the proxy in this case will allow for the engineered traffic to pass through. From Fig. 7 we can observe that even under this heavy load of incoming messages, the overall CPU rate is at around 30% which is between the thresholds used for controling the CPU usage.

Tab. 2 summarizes the results of this measurement. Out of nearly 330000 initiated calls around 50% of the calls were rejected. Out of the successful calls 92% were successfully terminated. 8% of the calls could not be terminated successfully as the sent BYE requests and their retransissions were dropped by the overloaded proxy.

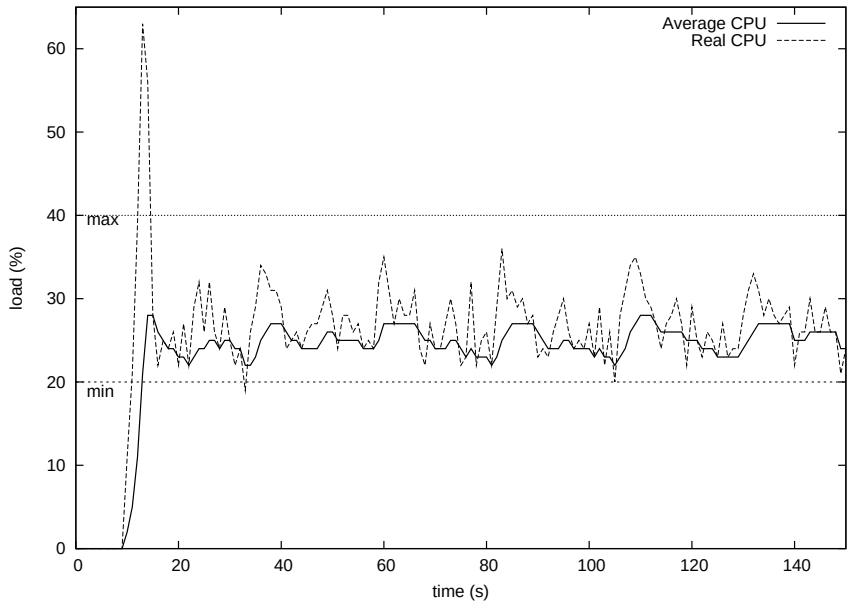


Figure 5: CPU performance of R-SOCA under BYE DoS attack

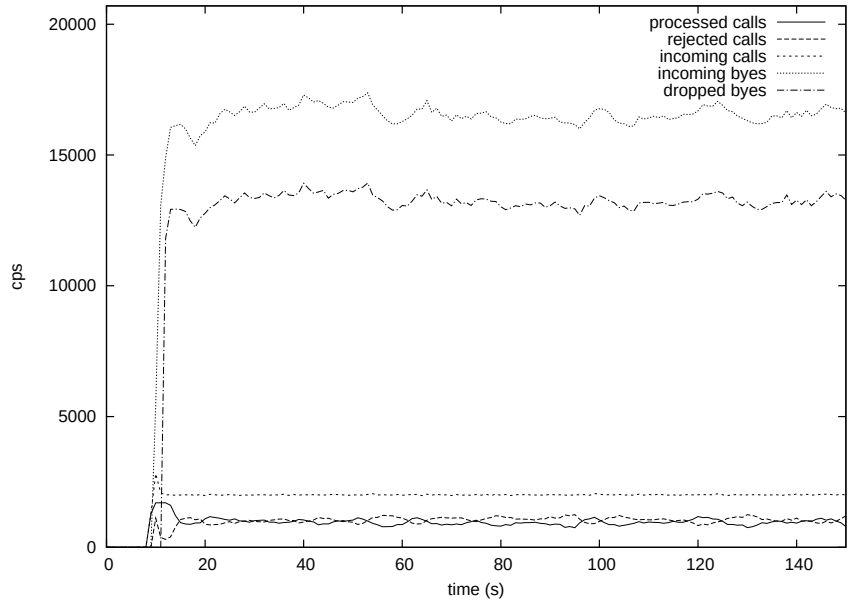


Figure 6: Performance of R-SOCA under BYE and INVITE DoS attack

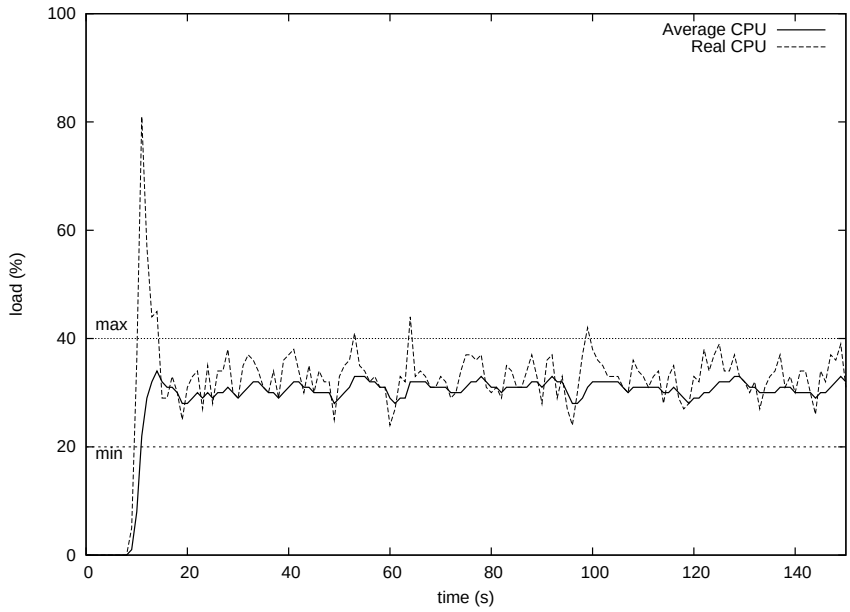


Figure 7: CPU Performance of R-SOCA under BYE and INVITE DoS attack

Sent Invites	Successful calls	Successful termination
363662	184290	168857

Table 2: Overall performance of R-SOCA under BYE and INVITE DoS attack

5 Summary and Future Work

In this paper we presented a novel scheme for dealing with overload situations in SIP proxies. The measurement results suggest that R-SOCA can achieve its goals and enable an overloaded SIP proxy to continue serving SIP traffic under high overload situations and denial of service attacks and still keep the resource usage at a pre-defined and expected level.

While already offering very promising results, R-SOCA will have to be extended to support additional features especially in the area of dynamic parameter setting. Currently, the thresholds used for determining the overload status of the SIP proxy are pre-defined and are set based on measurement of the system. Detecting and setting these thresholds dynamically would enhance the efficiency of R-SOCA and reduce the configuration overhead. Measuring the distribution of request types and setting the expected values for each request type dynamically, would further improve the DoS detection and enable R-SOCA to react faster to to such attacks.

Finally, besides optimizing the schemes, future work will investigate the performance of R-SOCA in real-live environments and under different usage scenarios.

References

[1] Signalling flows for the ip multimedia call control based on session initiation protocol (sip) and session description protocol (sdp). Technical specification group core network and terminals, 3rd Generation Partnership Project, 2007.

[2] A. Acharya, D. Kandlur, and P. Pradhan. Differentiated handling of SIP messages for VoIP call control. United states patent 20050105464, May 2005.

[3] S. Donovan and J. Rosenberg. Session Timers in the Session Initiation Protocol (SIP). RFC 4028 (Proposed Standard), Apr. 2005.

[4] M. Grossglauser and J.-C. Bolot. On the relevance of long-range dependence in network traffic. *IEEE/ACM Transactions on Networking*, 7(5):629–640, 1999.

[5] V. Hilt, I. Widjaja, D. Malas, and H. Schulzrinne. Session initiation protocol (sip) overload control. Internet Draft, Internet Engineering Task Force, Mar. 2007. Work in progress.

[6] J. Kuthan, S. Ehlert, and D. Sisalem. Denial of service attacks targeting a SIP VoIP iinfrastructure - attack scenarios and prevention mechanisms. *'IEEE Networks Magazine*, 20(5), Sept. 2006.

[7] E. Nahum, J. Tracey, and C. Wright. Evaluating SIP server performance. Research report RC24183, IBM T. J. Watson Research Center, Feb. 2007.

[8] E. C. Noel and C. R. Johnson. Initial simulation results that analyze sip based voip networks under overload. In L. Mason, T. Drwiega, and J. Yan, editors, *International Teletraffic Congress*, volume 4516 of *Lecture Notes in Computer Science*, pages 54–64. Springer, 2007.

[9] M. Ohta. Simulation study of sip signaling in an overload condition. In M. H. Hamza, editor, *Communications, Internet, and Information Technology*, pages 321–326. IASTED/ACTA Press, 2004.

[10] M. Ohta. Overload protection in a sip signaling network. In *International Conference on Internet Surveillance and Protection (ICISP'06)*, 2006.

[11] J. Rosenberg. Requirements for management of overload in the session initiation protocol. Internet Draft, Internet Engineering Task Force, Oct. 2006. Work in progress.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916.

[13] H. Schulzrinne, S. Narayanan, J. Lennox, , and M. Doyle. Sipstone - benchmarking sip server performance, Apr. 2002.

[14] C. Shen, H. Schulzrinne, , and E. Nahum. Sip server overload control: Design and evaluation. In *Conference on Principles, Systems and Applications of IP Telecommunications (IPTCOMM08*, Heidelberg, Germany, July 2008.

[15] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem. Denial of service attack and prevention on sip voip infrastructures using DNS. In *Principles, Systems and Applications of IP Telecommunications (IPTCOMM)*, NY, USA, July 2007.

Attacks on Message Stream Encryption

Billy Bob Brumley and Jukka Valkonen

Department of Information and Computer Science
Helsinki University of Technology
P.O.Box 5400, FIN-02015 TKK, Finland
{billy.brumley,jukka.valkonen}@tkk.fi

Abstract. Message Stream Encryption (MSE) provides obfuscation, data confidentiality, and limited authentication to BitTorrent clients. Although obfuscation of header and payload data was the main design goal of MSE, users understandably still expect data confidentiality and authentication from their BitTorrent clients. In this paper, we present numerous attacks on the MSE protocol itself, independent of clients. We then test many popular BitTorrent clients for vulnerability to these attacks, resulting in a number of serious vulnerabilities in popular clients. These results are timely and significant due to the high penetration rate of BitTorrent clients.

Key words: BitTorrent, peer-to-peer protocols, stream ciphers, man-in-the-middle attacks.

1 Introduction

In 2004, former CacheLogic now Velocix, a Cambridge, UK company, conducted a six month study in which they concluded that BitTorrent traffic accounted for roughly 33% of all Internet traffic [1]. Although such estimates vary and should be taken with a grain of salt, the proliferation of BitTorrent clients cannot be denied.

As such usage has an adverse affect on network performance, some ISPs have chosen to throttle BitTorrent traffic. In response to this, BitTorrent client developers designed Message Stream Encryption (MSE) [2]. The main design goal of MSE was indeed traffic obfuscation, with secondary goals of confidentiality and authentication.

Surprisingly, instead of using an existing, well-known public protocol as a basis (IPSec, for example), BitTorrent client developers themselves designed the MSE protocol from scratch. To this effect, in this paper we present a number of attacks on the MSE protocol, with vulnerabilities leading to complete key recovery and torrent fingerprint leakage. We apply these attacks to a number of individual clients for many different platforms. The results show that the MSE protocol has a significant number of weaknesses and the protocol description itself leaves too many details up to interpretation.

2 The BitTorrent Protocol

BitTorrent [3] is a protocol designed to distribute large files efficiently across networks. The idea of the protocol is to reduce the cost and burden of a single device when large files are being distributed. This is achieved by distributing the file into multiple small pieces which are then supplied by multiple clients. While devices download files using the BitTorrent protocol, they simultaneously start sharing the same pieces.

BitTorrent network consist of four entities: downloaders, seeders, web servers and trackers. In order to start sharing a file using the protocol, first a torrent file is created. The file includes all information needed to download all the pieces of the shared file. Basically, this means tracker information, the total amount of pieces the file is shared into and hash values of all the pieces. These hash values are used to verify the integrity of the file after the pieces are downloaded. In addition, the file includes a hash file called InfoHash which is computed from the other metadata in the torrent file and thus can be used to identify the specific torrent. In addition to giving the torrent file for users to download, a torrent tracker needs to be informed about the file. The

basic task of the tracker is to keep track of all the seeders of a file and thus help the downloading clients in communication with each other.

The torrent file is uploaded on a web server to be distributed to downloaders. When a user would like to download a file using BitTorrent, he first contacts the web server to download the torrent file. (Message 1 of Figure 1). The file can be downloaded in a number of ways, normally via HTTP or HTTPS. After downloading the torrent file, the BitTorrent client contacts the specified tracker to get the current list of seeders seeding the file to be downloaded. (Message 2 in Figure 1).

After receiving the list of seeders, the client contacts them one at a time (Figure 1: 3a,3b,3c) to download a specific piece of the file the seeder is providing. After downloading a piece (or multiple pieces), the client is able to verify the integrity of the file using the information from the torrent file.

If compared to downloading by traditional means, the speedup of BitTorrent protocol comes from spreading the file over multiple devices. All peers in the BitTorrent network need to seed only small pieces over multiple TCP connections, whereas in traditional web downloading the large files are distributed by single serves over single TCP connections. In addition, in HTTP downloading the file is downloaded in sequential manner, whereas in BitTorrent the pieces are downloaded in randomized order.

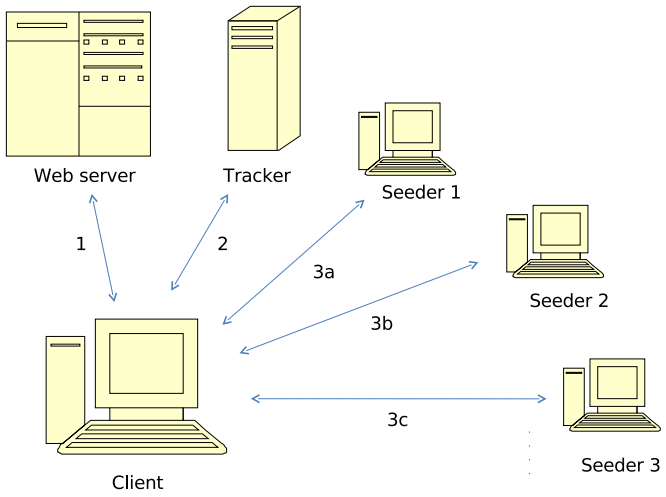


Fig. 1. BitTorrent operation

2.1 Torrent Handshake

We briefly describe the handshake of the BitTorrent protocol itself [3]; this takes place in the layer above MSE. The handshake is the first message transmitted between clients and is generally used to ensure that two clients are indeed about to share the same file. The handshake is 49+PStr bytes long and consists of the following fields (in order):

- PStrLen: 1 byte. The length of the PStr field.
- PStr: PStrLen bytes. Protocol identifier; for torrents, "BitTorrent protocol" and PStrLen=19.
- Reserved: 8 bytes. Initially all-zero, used to modify the protocol behavior.
- InfoHash: 20 bytes. The SHA1 hash of the info key in the metainfo file; unique identifier for a single torrent.
- PeerID: 20 bytes. A unique string identifier for the client. In practice, two bytes are used to identify the client software, four bytes for the client software version, then random numbers.

- g: Generator; with MSE $g = 2$ generates $\mathbb{Z}_p^\times$ with 768-bit safe prime $p = 2q + 1$.
- a, b: Diffie-Hellman exponents, MSE recommends 160 random bits.
- S: Negotiated Diffie-Hellman key, $S = (g^a)^b = (g^b)^a = g^{ab}$.
- PadA, PadB: Random padding of length 0–512 bytes.
- H: Hash function, SHA-1 which outputs 20 bytes. Literal strings are enclosed in ' '.
- SKey: Weak shared secret key; for BitTorrent, SKey = InfoHash of size 20 bytes.
- E: Encryption, RC4 with key H('keyA',S,SKey) for A and H('keyB',S,SKey) for B.
- $\hat{E}$: Negotiated encryption; currently either plaintext or the continuation of the above encryption.
- VC: Verification constant, currently 8 bytes of `0x00`. Used for synchronization.
- CryptoProvide: 4 byte field denoting which encryption primitives are supported; currently `0x01` = plaintext and `0x02` = RC4. Supported methods are combined with XOR.
- CryptoSelect: 4 byte field denoting the chosen encryption primitive, using the same values as above.
- len: Length function, outputs the byte length of the input as 2 bytes.
- PadC, PadD: Zero-valued padding of 0–512 bytes.
- IA: Initial payload; for BitTorrent, A's portion of the BitTorrent handshake.

1. A → B: g^a, PadA
2. B → A: g^b, PadB
3. A → B: H('req1', S), H('req2', SKey) ⊕ H('req3', S), E(VC, CryptoProvide, len(PadC), PadC, len(IA)), E(IA)
4. B → A: E(VC, CryptoSelect, len(padD), padD), $\hat{E}$(Payload Stream)
5. A → B: $\hat{E}$(Payload Stream)

Fig. 2. Message Stream Encryption Protocol handshake

If a seeder client receives a handshake with an InfoHash they are not serving, they would be expected to drop the connection.

3 Message Stream Encryption

Message Stream Encryption [2] is designed to provide security features to the BitTorrent protocol at a lower layer by acting as a wrapper. The main goal of the protocol is to provide obfuscation for the data streams. This is done to prevent passive eavesdroppers from being able to recognize the protocol that is used. Recently, this has become an issue as some Internet service providers have started to monitor the data streams and in some cases limit the bandwidth of clients using BitTorrent.

In addition to obfuscation, secondary goals of the protocol appear to be providing some level of confidentiality and authentication of the communicating peers; this is evident from the cryptographic primitives being used.

The handshake of the protocol is depicted in Figure 2. The protocol uses Diffie-Hellman key exchange [4] to create a session key (surprisingly, not one of the numerous standard methods for authenticated key exchange [5–7] since a pre-shared secret is available). In addition, a weak shared key is used to provide some level of authentication to the key exchange. The InfoHash value transmitted in the torrent file is used for this weak shared secret, in this case it's value being similar to a pre-shared key in the context of peer-to-peer networks.

After negotiating the Diffie-Hellman key, the rest of the handshake is encrypted using RC4 [8], with the first 1024 keystream bytes discarded to defeat the attacks in [9, 10]–clearly some level of confidentiality (say, against those not participating in the torrent) is desired. The RC4 key is based on both the Diffie-Hellman key and the weak shared secret, thus a traditional man-in-the-middle attack will not be successful without the weak shared secret. The payload is either encrypted using RC4 or sent in plaintext. The desired payload delivery method is negotiated in Messages 3 and 4. In Message 3, the initiator sends the method(s) it is willing to use in CryptoProvide and the responder selects the desired method in CryptoSelect.

Note also that MSE was designed as a general payload delivery protocol, not specific to BitTorrent. When looking for weaknesses in MSE, it is thus important to consider what the intended payload is. Weakness in MSE need not be connected to BitTorrent in any way.

3.1 Security and MSE

The security of peer-to-peer networks like BitTorrent inherently has different goals than other networks like GSM or wireless LANs. The driving force behind P2P networks is the openness and ability to distribute and connect. In fact, the original creator of the BitTorrent protocol, Bram Cohen, was opposed to the widespread adoption of MSE, concerned that it would cause too much incompatibility between clients.

We quote directly from the MSE protocol specification [2] describing the design methodology of MSE:

> It is also designed to provide limited protection against active MITM attacks and portscanning by requiring a weak shared secret to complete the handshake. You should note that the major design goal was payload and protocol obfuscation, not peer authentication and data integrity verification. Thus it does not offer protection against adversaries which already know the necessary data to establish connections (that is IP/Port/Shared Secret/Payload protocol).

We consider what the protocol states it is designed to protect against. The payload protocol is clearly the BitTorrent Peer Wire (TCP) protocol in this case. To carry out an attack, an attacker would presumably have to know (or have a way to get) the target connection details such as IP and port. Thus the only seemingly interesting case to attack, and for very logical reasons, is when an attacker does not know the shared secret. This shared secret is considered "weak" as the privacy differs greatly from what would normally be expected of a shared secret. In this case, the weak shared secret being the InfoHash of the file means that if you know what torrent you want to participate in, you are able to, and thus such an attacker is free to unleash a plethora of attacks. Such a simple protocol cannot hope to protect against this.

Thus, in this paper we restrict to the case where the attacker does not know the weak shared secret. This is not at all unreasonable, as tracker data can (and for MSE to provide confidentiality, clearly should) be exchanged over SSL, so the simple attack target would be unveiling the data being shared through a seemingly secured BitTorrent protocol using MSE.

We are thus not attacking in any way the obfuscation portion of MSE. The attacks considered here should be considered targeted attacks that tell something about the payload. This would not be useable by an ISP on a large scale to throttle BitTorrent traffic, but could be used to target a specific user. For example, a law enforcement agency might want to know what payload a user is transmitting or receiving (say, for copyright infringement issues). A user would reasonably expect MSE to be able to defend against such attacks.

4 Message Stream Encryption Weaknesses

In this section, we present some major weakness of MSE, independent of the client implementations. These attacks are applied to a number of BitTorrent clients in Section 5.

4.1 Lack of Message Authentication

As there is no message authentication present on the messages sent in the MSE handshake, attackers are free to modify any of the messages sent in an undetectable manner. With RC4 being a stream cipher, it becomes incredibly simple for attackers to make predictable changes in resulting plaintext after decryption. We now show how attacker can modify the messages

sent in the MSE handshake to trick the devices into downgrading the encryption into plaintext, leading easily to recovery of the weak shared secret (SKey, InfoHash).

The stream cipher RC4 produces ciphertext by combining a pseudorandom sequence of bytes with plaintext using the XOR operation. The attack works simply by flipping bits of the CryptoProvide field by using the XOR operation on the ciphertext, as shown in Figure 3. In order to be able to modify the fields in practice, the attacker needs to be able to synchronize with the handshake and locate the CryptoProvide value in Message 3 of Figure 2 to get the devices to downgrade the encryption. An attacker can normally accomplish this as the messages of the handshake are usually sent in separate packets. After the attacker has located the correct message, finding the CryptoProvide field is easy: the lengths of the fields before the desired fields are known from the specification. The attacker then makes the desired changes using the XOR operation, and is able to downgrade the encryption from RC4 to plaintext. For a client desiring the highest level of security, only a value of CryptoProvide=0x02 would be sent, meaning the client only provides RC4. The limited number of possible values makes it simple for an attacker to achieve the desired effect through trial and error.

A → B:	H('req1', S)	H('req2', SKey) ⊕ H('req3', S)	VC	CryptoProvide	...
	20 bytes	20 bytes	0000000000000000 ⊕ $b_0 \dots b_7$	00000002 ⊕ $b_8 \dots b_{11}$	...
Attacker:	↓	↓	↓	⊕ 00000003	...
B:	—	—	⊕ $b_0 \dots b_7$ = 0000000000000000	⊕ $b_8 \dots b_{11}$ = 00000001	...

Fig. 3. Encryption downgrade attack on MSE.

Having now received a CryptoProvide value of plaintext only, B responds with Message 4 of Figure 2. The last portion of this message is the payload in plaintext, which is B's portion of the BitTorrent protocol handshake as described in Sec. 2.1; this includes the weak shared secret (SKey, InfoHash). Having recovered the weak shared secret, the attacker now knows what torrent is being shared and can, for example, run a traditional man-in-the-middle attack to recover actual bytes being sent with the torrent.

It is also entirely possible that B rejects providing plaintext only. However, as the protocol does not provide any kind of message integrity checking, it is up to the implementation of client B to drop the connection if plaintext is selected even though encryption is required. Results on implementations are described in Section 5.

Note that such bit flipping as described above is also possible with the CryptoSelect field in Message 4 of Figure 2. However, the attacker is less likely to be successful in this case; the method of encryption has already been chosen by B in the last portion of Message 4 and will be evident when received by A, who will be expecting B's portion of the BitTorrent protocol handshake.

The attack can be prevented quite easily. The protocol should implement some kind of integrity checking. This could be achieved using for example Message Authentication Codes (MAC) (see for example [11] or Cyclic Redundancy Check (CRC) (e.g. [11]) techniques. As the protocol already uses a (weak) shared secret, implementing a method based on a MAC would be quite easy. For example Wired Equivalent Privacy (WEP) [12] uses RC4 cipher together with Cyclic Redundancy Check CRC-32.

4.2 Keystream Leakage

Many of the fields in the MSE handshake have a large number of bytes but take either one or a small number of values. Ordinarily, this would not be a significant problem; however, the zero-valued variable length padding causes immediate leakage of a significant amount of keystream bytes. Again, this does not seem like a big issue–unless the keystream is reused. Below we present an attack that exploits the zero-valued padding when the keystream is reused.

In two runs of the MSE handshake for the same torrent, consider two values of Message 3 of Figure 2 denoted c_1 and c_2 using the same keystream. The length of PadC is a random value from 0 to 512, thus with all probability one of c_i is longer than the other; assume without loss of generality that c_1 is longer than c_2, and we denote the length of the PadC values as i and j respectively. The MSE protocol description states of PadC and PadD: "For padding-only usage in the current version they should be zeroed." We examine the construction of the c_i in Figure 4. We can see from the very first message that because of the fixed values of many of the fields, we recover many keystream bytes b, and with $j < i$ most noteably $b_{14+j} \dots b_{14+i-1}$. With all probability, this leads to the instant recovery of IA from c_2 by just XORing the fixed known keystream bytes. Again, this initial payload consists of A's portion of the BitTorrent protocol handshake, and thus contains the weak shared secret (SKey, InfoHash) which can be recovered as long as $j + 48 \leq i$ (see Sec. 2.1). This already holds with high probability for two sessions, but can be repeated as many times as necessary to obtained the required result.

	VC	CryptoProvide	len(PadC)	PadC		len(IA)	IA
$c_1 =$	0000000000000000	00000002	????	00 ... 00	00 ... 00		
	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	...	...
	$b_0 \dots b_7$	$b_8 \dots b_{11}$	$b_{12}b_{13}$	$b_{14} \dots b_{14+j-1}$	$b_{14+j} \dots b_{14+i-1}$		
$c_2 =$	0000000000000000	00000002	????	00 ... 00		????	?? ...
	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$	$\oplus$
	$b_0 \dots b_7$	$b_8 \dots b_{11}$	$b_{12}b_{13}$	$b_{14} \dots b_{14+j-1}$		$b_{14+j}b_{15+j}$	$b_{16+j} \dots$

Fig. 4. MSE Secret key recovery attack by keystream reuse.

Forcing Keystream Reuse in Practice With the attack in Figure 4, the attacker plays the role of B and thus would need some way to force reuse of the keystream. At first glance, the above attack does not seem very practical as the only way to force keystream reuse is when the same RC4 key is reused. Above, it was assumed that the same torrent was being accessed in both runs of the MSE handshake; thus, the only way the keystream would be reused is if the negotiated Diffie-Hellman key S was the same in both runs. We give two examples of how an attacker might accomplish this; results of running these attacks on different BitTorrent client implementations are given in Section 5.

Keystream reuse using trivial subgroups. With $p = 2q+1$ a safe prime, the only small subgroups of $\mathbb{Z}_p^\star$ are the trivial ones; that is, $\{1\}$ of multiplicative order 1 and $\{-1, 1\}$ of order 2. The Diffie-Hellman key can be confined to one of these trivial subgroups by raising the public keys sent to the power of $2q$ or q, respectively [13]. A simple example of this attack follows. A attempts to open a connection to B; the attacker is in the middle and impersonates B, receives Message 1 and sends the public key value 1 in response, records Message 3 from A and drops the connection. The process is repeated again, either for the same A or different A. The Diffie-Hellman key $\mathsf{S} = 1$ is the same in both runs of the MSE handshake, and the attack in Figure 4 is most likely successful. Unlike the MSE specification, most standards explicitly state to validate public keys; for example, the Wireless USB association models specification [14].

Keystream reuse using discrete logs. Keystream reuse can also be forced if computing discrete logs of the public keys sent in the Diffie-Hellman key agreement is possible. For example, A sends $g^{a_1} \mod p$ and the attacker sends $g^{b_1} \mod p$, records Message 3 from A and drops the connection. Now A sends $g^{a_2} \mod p$. Assume the attacker can somehow compute a_1 and a_2. The Diffie-Hellman negotiated key for the first session was $\mathsf{S}_1 = g^{a_1 b_1} \mod p$. To force keystream reuse in the second session, the attacker chooses b_2 such that $a_1 b_1 \equiv a_2 b_2 \pmod{\mathrm{ord}(g)}$ holds. When the attacker sends $g^{b_2} \mod p$ in the second session, the negotiated key $\mathsf{S}_2 = g^{a_2 b_2} \mod p = \mathsf{S}_1$ and the same keystream will be used in the second session. The textbook example of how computing discrete logs could be easy for a given client implementation is poor pseudo random number generation—for example, seeding with a 32-bit value or repeated seeding based on the system clock or CPU counter.

4.3 Torrent Fingerprint Leakage

As the protocol uses the weak shared secret SKey as part of the encryption key, the devices communicating must be able to negotiate which key to be used. Clients seed many torrents concurrently, thus determining which SKey to use in MSE is tricky. When the communication is initiated, the devices presumeably share the same SKey, the seeder simply does not know which key the downloader is using (and thus what torrent is being accessed). In the current implementation, the client tells the seeder which key to use in Message 3 in Figure 2 by sending a hash of the SKey; we denote this as the *torrent fingerprint*. An active man-in-the middle is able to obtain this fingerprint using XOR as it knows the negotiated Diffie-Hellman key S. As there is no randomness added to this hash, the torrent fingerprint (as the name implies) is unique per torrent; a good analogy is to that of comparing salted password hashes (e.g. classical Unix password authentication) to password hashing with no salting.

The fingerprint leakage may lead to recovery of the SKey and/or the actual file being downloaded. As the SKey values are distributed within the torrent files, which are distributed to clients using normal web servers, the attacker can, after seeing a hash of some SKey, browse through the Internet for torrent files and compute SHA-1 hashes of them offline. At some point, the attacker may find the same fingerprint it has already seen. This could be considered a brute-force attack if torrents are taken at random, or a dictionary attack if an attacker has a fixed set of torrents they wish to check the torrent fingerprint against. In either case, the keyspace is very limited and all the keys are distributed on the Internet. For example, the administrator of a torrent website serving torrent files could easily, with one SQL query, generate all the torrent fingerprints of every torrent available from its server (and even publish this list). Adding per-session randomness to this hash could prevent such an attack (at the significant cost of seeding client performance), but still does not prevent the calculation of the hash values online even when randomness is included (although this is somewhat less of a threat).

This leakage also leads to the following security issue: even though the attacker is not able to directly tell which torrent is being downloaded, it can say if the torrent is the same as downloaded by another client in another instance as the SKey and thus torrent fingerprint is unique for each torrent file.

The situation is illustrated in Figures 1 and 5. In Figure 1 the client is downloading pieces from multiple seeders. As for all these seeders the InfoHash is the same and the same hash is sent in messages 3a, 3b and 3c, the attacker is able to tell that in all these connections the same file is being downloaded. Figure 5 depicts a situation where two separate swarms have been created. If the attacker is active in at least one pair of devices sharing and downloading a piece of a file in both of the swarms, it is able to tell if the same file is being downloaded by simply comparing the torrent fingerprints.

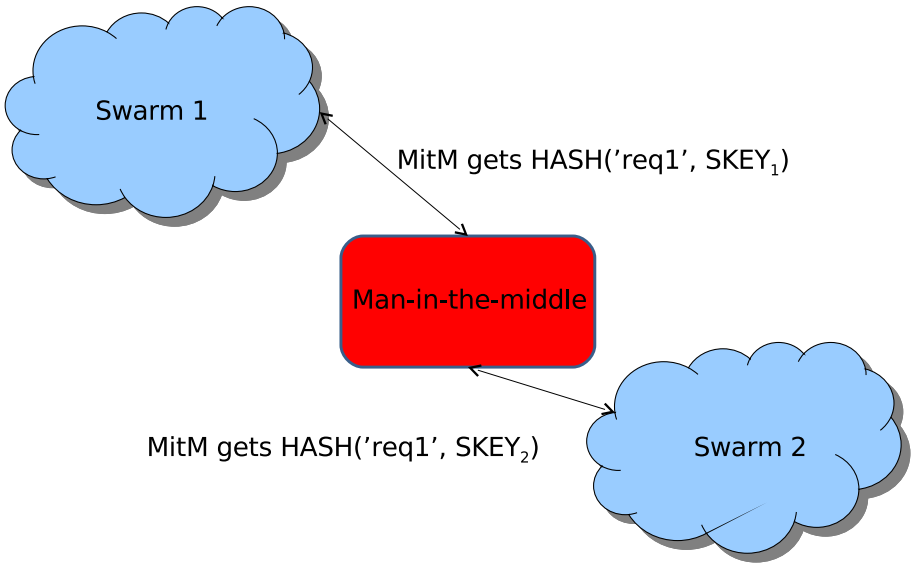


Fig. 5. Torrent fingerprint by man-in-the-middle in two swarms.

5 Vulnerabilities on Implementations

As we were interested in seeing how the real implementations handle the previously described security weaknesses, we implemented our own client capable of acting as both a client and a server in MSE. This was done as many BitTorrent clients are closed source; even with open source, understanding the implementation (and its flaws) can be quite difficult. By implementing a custom client, we were able to obtain full control and a clear idea of how different clients have implemented MSE. Next, we list some properties of tested clients.

5.1 Popular Clients

In 2006-7, Digital Music News conducted a study on the penetration of individual BitTorrent clients [15]. Statistics were collected from users voluntarily using an online virus scan site. Over 1.7 million PCs were examined. Table 1 (left) lists the results; (right) shows P2P software marketshare. The data was gathered by PC Pitstop as late as January 2008 [16], where over one million PCs were examined. We can conclude that μTorrent has by far the greatest market penetration for a BitTorrent client, at least for Windows.

Rank	Client	Installations (%)
1.	μTorrent	5.56
2.	BitTorrent	2.28
3.	Azureus	2.11
4.	Bitcomet	1.89
5.	Bitlord	1.27

Client	Marketshare (%)
LimeWire	37.19
μTorrent	13.51
BitTorrent	5.31
Ares	4.35
Emule	3.69
BitComet	3.64
Azureus	3.05
Other	29.21

Table 1. Popularity of BitTorrent clients.

The clients we tested for vulnerabilities are listed below. From this list, the degree of support for different MSE options ranges greatly.

BitTorrent 6.0.2 (Build 8388). Windows, closed source. This is the "official" BitTorrent client. Through Options, Preferences, BitTorrent, there are three options for Protocol Encryption,

Outgoing: Disabled, Enabled, and Forced. There is a checkbox for "Allow incoming legacy connections".

µTorrent 1.7.7 (Build 8179). Windows, closed source. Through Options, Preferences, BitTorrent, there are three options for Protocol Encryption, Outgoing: Disabled, Enabled, and Forced. There is a checkbox for "Allow incoming legacy connections".

BitComet 0.99. Windows, closed source. Under Options, Options, Advanced, Connection, there are three options for Protocol encryption: Auto Detect, Always, and Disable.

Azureus 3.0.5.0. Multi-platform, Java, open source. Through Tools, Options, Connection, Transport Encryption, there are two options for Minimum encryption level: Plain and RC4. "Require encrypted transport" enables and disables MSE. There are two checkboxes for fallback options to allow incoming and/or outgoing unencrypted connections, as well as an option to support the cryptoport tracker extension.

KTorrent 2.1. Linux, C/C++, open source. Through Settings, Configure KTorrent, General, under Encryption there are two checkboxes for "Use protocol encryption" and "Allow unencrypted connections".

5.2 Results

A summary of the attack results is given in Table 2; we provide details of the individual attacks below.

Encryption downgrade. We tested the clients in two directions: the custom client acting as party A (initiating connections, client) and as party B (receiving connections, server). In both scenarios, the real client was set to allow only encrypted (i.e. RC4) connections. Our custom client then attempted to either provide or select (ignoring the other party's CryptoProvide field) plaintext only; according to the MSE specification, it is up to the client to drop the connection in these cases, although for this attack scenario, as previously mentioned it would be much more convenient to prevent the attack with the use of a MAC. The clients µTorrent and BitTorrent were found to be vulnerable; we were unable to find a difference between the "Enabled" and "Forced" for the "Outgoing" option of Protocol Encryption. KTorrent, Azureus, and BitComet all rejected selecting an unsupported encryption method, and hence were not found to be vulnerable.

Keystream reuse, trivial subgroups. To test for this vulnerability, three different values were sent as the public key to the real clients: values of 1, $p-1$, and 0. If the MSE handshake was successful, forcing keystream reuse via subgroup confinement is possible for that particular client. As the MSE specification does not specify to drop connections when receiving such values, it is up

Client	Encryption downgrade	Keystream reuse, subgroups	Keystream reuse, discrete logs	SKey Recovery	Fingerprint leakage
BitTorrent	√	√	×	√	√
µTorrent	√	√	×	√	√
BitComet	×	×	×	×	√
Azureus[a]	×	×	×	√	√
KTorrent	×	√	√	√	√

Table 2. Summary of client vulnerabilities; √ =Vulnerable, × =Not Vulnerable.

[a] In sessions initiated by an Azureus client (e.g. acting as A), the BitTorrent handshake is sent in plaintext first—thus the shared secret is instantly revealed; we cannot explain such behavior.

```
BigInt BigInt::random()
{
static Uint32 rnd = 0;
if (rnd % 10 == 0) // reset every 10 calls
{
        TimeStamp now = bt::GetCurrentTime(); // seeding on system time
        srand(now); // 32-bit seed
        rnd = 0;
}
rnd++;
Uint8 tmp[20];
for (Uint32 i = 0;i < 20;i++)
        tmp[i] = (Uint8)rand() % 0x100;
return BigInt::fromBuffer(tmp,20);
}
```

Fig. 6. KTorrent 2.2.7, excerpt from `bigint.cpp` for random number generation.

to the implementation (thus these keys are valid according to the MSE specification, but it is still entirely possible that the implementation has foreseen such attacks). Many cryptographic packages perform such public key validation automatically, and a client that does so is clearly less likely to be vulnerable. Only two of the clients examined are open source; KTorrent's public key portion of the MSE implementation is done using a simple BigInteger package, and thus it is easy to identify the vulnerability. The clients μTorrent, BitTorrent, and KTorrent were all found to be vulnerable, while Azureus and BitComet both dropped the connection.

Keystream reuse, discrete logs. As mentioned above, only two of the clients are open source; we only examined these two clients for this particular attack. Azureus uses the canned Java methods for random number generation, hence we did not attempt any attacks on computing discrete logs with Azureus. However, KTorrent uses the standard C `srand` function to seed the pseudo-random number generator (PRNG). The seed provided by the current system time, reset every 10 calls. The function `srand` casts the provided seed as a 32-bit integer, and therefore it has a total entropy of $10 \cdot 2^{32} \approx 2^{35}$; see Figure 6, with the vulnerabilities highlighted in red. An attacker can easily compute all of these values offline, or attempt to run a more intelligent attack by guessing the seed based on the time.

Torrent fingerprint leakage. As this vulnerability is inherent to the MSE protocol, any client implementing MSE is clearly vulnerable.

6 Conclusion

During the past few years, the BitTorrent protocol has gained a large interest between users of the Internet for payload and content delivery, accounting for a significant portion of network traffic. The MSE protocol was designed to obfuscate such data, as well as provide limited confidentiality and authentication between peers.

In this paper, we have identified a number of significant weaknesses in the MSE protocol. We emphasize that these protocol weaknesses are not useful for ISPs wanting to throttle BitTorrent bandwidth on their networks, but lead to frighteningly feasible targeted attacks by ISPs or law enforcement agencies to reveal seemingly confidential data. We have shown that MSE does not provide a good defense against such targeted attacks.

At a minimum, we recommend the following changes to the MSE specification:

1. Add a Message Authentication Code (MAC) to the messages;
2. Drop connections if public keys (say K_A) do not satisfy $1 < K_A < p - 1 \pmod{p}$;

3. Remove or randomize padding that is encrypted, specifically the PadC and PadD values.

A solution to the torrent fingerprint leakage needs further investigation, taking efficiency, practicality, and security into account.

In conclusion, MSE still does a good job at data obfuscation as a whole. However, users expecting anything other than obfuscation, such as data confidentiality or authentication, cannot rely on MSE as the attacks presented here have shown.

Acknowledgments

The experiments appearing herein were carried out during the course "Special Course in Practical Security of Information Systems" at the Telecommunications Software and Multimedia Laboratory (TML), Helsinki University of Technology. We thank those that provided feedback, in particular the responsible lecturer Sami Vaarala.

Additionally, we thank Kaisa Nyberg for comments.

References

1. Andrade, N., Mowbray, M., Lima, A., Wagner, G., Ripeanu, M.: Influences on cooperation in bittorrent communities. In: P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems, New York, NY, USA, ACM (2005) 111–115
2. MSE: Message stream encryption protocol. Azureus Wiki (January 2006) http://www.azureuswiki.com/index.php/Message_Stream_Encryption.
3. Cohen, B.: The BitTorrent protocol specification (January 10, 2008) http://www.bittorrent.org/beps/bep_0003.html.
4. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Information Theory **IT-22**(6) (1976) 644–654
5. IEEE: Standard specifications for password-based public-key cryptographic techniques. Draft IEEE P1363.2 / D26, Institute of Electrical and Electronics Engineers, Inc. (September 2006)
6. Bellovin, S.M.; Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. IEEE Computer Society Symposium on Research in Security and Privacy (4-6 May 1992) 72–84
7. Jablon, D.P.: Strong password-only authenticated key exchange. SIGCOMM Comput. Commun. Rev. **26**(5) (1996) 5–26
8. Thayer, R., Kaukonen, K.: A stream cipher encryption algorithm 'arcfour'. Internet Engineering Task Force (July 1999) http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt.
9. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Fast Software Encryption. Volume 2355 of Lecture Notes in Comput. Sci., Springer Berlin / Heidelberg (2002) 87–104
10. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Selected areas in cryptography. Volume 2259 of Lecture Notes in Comput. Sci. Springer, Berlin (2001) 1–24
11. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
12. IEEE Computer Society: IEEE Standard for Information Technology - Telecommunications and information exchagne between systems- Local and metropolitan area networks- Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications http://standards.ieee.org/getieee802/download/802.11-2007.pdf/.
13. van Oorschot, P.C., Wiener, M.J.: On Diffie-Hellman key agreement with short exponents. In: Advances in cryptology—EUROCRYPT '96. Volume 1070 of Lecture Notes in Comput. Sci. Springer, Berlin (1996) 332–343
14. WUSB: Association Models Supplement to the Certified Wireless Universal Serial Bus Specification - Revision 1.0 (2006) http://www.usb.org/developers/wusb/wusb_2007_0214.zip.
15. Digital Music News: Digital media desktop report, third quarter (2007)
16. TorrentFreak: Filesharing report shows explosive growth for uTorrent. http://torrentfreak.com/p2p-statistics-080426/ (April 2008)

Towards a quantitative assessment of security in software architectures

Artsiom Yautsiukhin[2*] Riccardo Scandariato[2] Thomas Heyman[2] Fabio Massacci[1]
Wouter Joosen[2]

[1] - University of Trento, TN, Italy
[2] - Katholieke Universiteit Leuven, Belgium

Abstract

Software patterns are key building blocks used to construct the architecture of a software system. Patterns also have an important role during the architecture assessment phase, as they represent the design rationale, which is central to evaluation. This work presents a quantitative approach to assess the security of a pattern-based software architecture. In particular, security patterns are used to measure to what extent an architecture is protected against relevant security threats. To this aim, threat coverage metrics are associated to security patterns and an aggregation algorithm is proposed to compute an overall security indicator. The proposed approach helps in comparing design alternatives and choosing the best candidate.

1 Introduction

The problem of assessing the level of security of a software system is still largely open. This is because the complexity of todays systems is becoming cumbersome. Systems are growing bigger and are often the result of tangled composition. They are more interconnected and more heterogeneous in the technologies they adopt. In this complex context, the security discipline lags behind in the area of quantitative assessment methods. Most of the literature focuses on the low end of the spectrum, i.e., on assessing the security posture after deployment.

The software architecture discipline is able to counteract the mentioned limiting factors factors to the assessment of security. It is well known that the software architecture is the key place where software qualities (including security) are embedded in a software system. It is also recognized that the software architecture provides a *manageable* abstraction, in terms of size and complexity, to perform an effective (often critical) quality assessment on the system in a holistic way. In this respect, some approaches exist, but they are qualitative in nature [3]. To the authors' knowledge, no approach exist that quantitatively assesses security at the software architecture level.

Software patterns are key building blocks used to build the architecture of a software-intensive system [1]. Patterns also have an important role during the architecture assessment phase, as they represent the design rationale, which is central to evaluation [3]. The main contribution of this paper is to provide a quantitative approach to assess the security of pattern-based software architectures. In particular, security patterns are leveraged to measure to what extent an architecture is protected with respect to relevant security threats. To this aim, threat coverage metrics are associated to security patterns and an aggregation algorithm

*The work by Artsiom Yautsiukhin was partly supported by the EU-IST-IP-SERENITY and IST-FP7-IP-MASTER projects

is proposed to compute an overall security indicator for the software architecture that instantiates the patterns. The proposed approach helps in comparing design alternatives and choosing the best candidate. In the context of this work, the fit criterion for the best candidate is represented by the best protection against the most dangerous threats.

The rest of this paper is structured as follows. First, in Section 2 we introduce the case study we use as an example throughout the paper. Then, in Section 3 we sketch the results of our previous works on which this paper is based. Section 4 describes how to determine the threat coverage of a security pattern. In Section 5 we show how the values can be aggregated and present an algorithm for that purpose. The results are validated on the case study in Section 6. Section 7 discusses the proposed approach. Related work (Section 8) and concluding remarks (Section 9) close the paper.

2 Case study

We use the following case study as a running example to illustrate our approach. An on-line publishing system has to be designed and the system should be able to receive news articles from *journalists* and advertisements from *advertisers*, allow *editors* to design upcoming editions of the newspaper using the news and advertisements, and provide access to the editions to *customers*. Obviously, next to providing the primary functionalities, there are a number of security-related issues which have to be encompassed in the system design, such as availability of the service, integrity of news items, and authentication of actors (journalists, customers, and so on).

3 Background

Previous work has studied the existing security pattern landscape and has shown how patterns can be used to fulfill security requirements in software design [4, 14, 18]. In this section, some of the previous results that are relevant to the present work are outlined.

Reuse of well-know and time-tested solutions (i.e., patterns) is a widely accepted software engineering practice. Patterns are even more relevant in the *secure* software engineering discipline because of the recognized principle of reusing community resources to avoid reinventing ad-hoc solutions from scratch.

Unfortunately, not all published security patterns are of straightforward use to the software developer, as they do not belong to the proper level of abstraction and lack a sufficiently detailed description. In order to address this problem, a preliminary list of patterns[1] which are directly usable to the software engineer (so-called *core patterns*), have been collected in [18]. The security patterns in this repository were used in selection of patterns for the case study presented in this paper.

To make optimal use of the pattern inventory, a methodology is needed that supports the selection of the right set of patterns for the (security) requirements at hand. Such methodology is presented in [14]. It enables the traceability of selection decisions as far as patterns and requirements are concerned. Such a relation is needed to perform the evaluation of the resulting pattern-based architecture, i.e., to determine how well the requirements are satisfied with the instantiation of a specific set of patterns.

The link from security requirements to security patterns is not obvious, as requirements are domain-dependent while patterns (by definition) are not. To solve this mismatch, security objectives are used as 'intermediaries'. Indeed, security objectives provide an abstraction that is conceptually close to requirements. However, objectives are domain-independent and can be completely enumerated. This allows attaching patterns to security objectives independently from the domain of their application.

[1] The remainder of this paper focuses on security patterns only. We refer to them with 'pattern' for brevity.

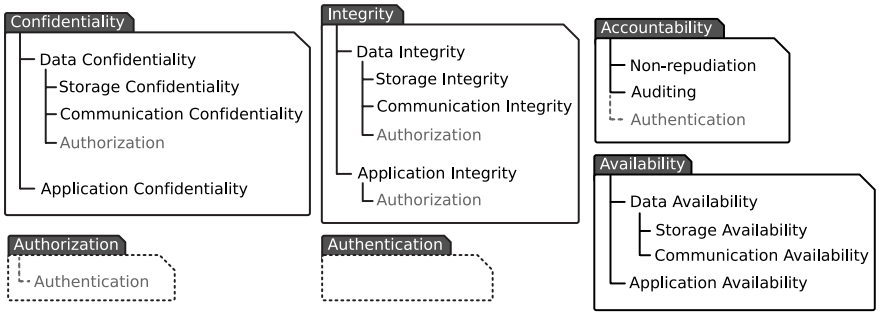


Figure 1: Decomposition of security objectives

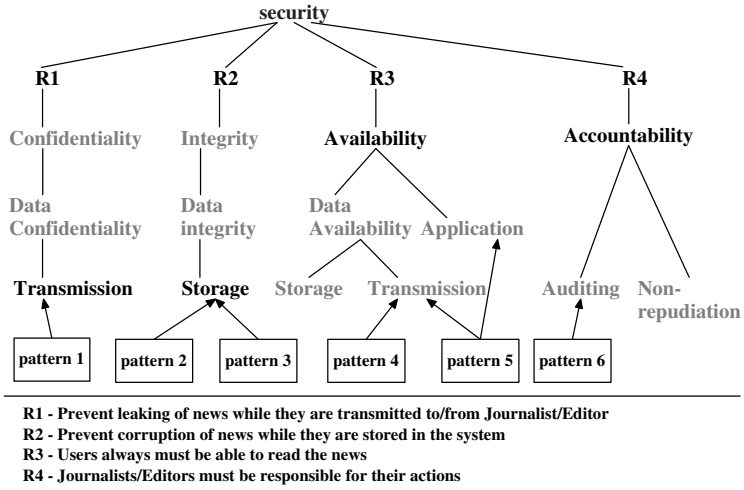


Figure 2: Requirements Tree for the running example.

We use the security *Objectives Decomposition* that is shown in Figure 1, which is based on [14]. There are four top security objectives (authentication and authorization are auxiliary objectives). These four objectives are often used to describe security as a whole [16, 8]. The high level objectives are, in turn, decomposed according to the areas of their applicability (e.g., application, data in transmission, stored data). This level of decomposition helps to sort patterns according to what (and in what context) they should protect. Further decomposition requires more details about the design of the system, which are not available at this high level of the analysis.

Each security requirement (or group of related requirements) has a motivation and rationale that lead to its definition. In general, it is straightforward to interpret such a rationale as a security objective associated to the requirement. By creating an association between requirements and objectives in the problem domain and between objectives and patterns in the solution domain, a link is establish that can be leveraged in two directions. Top-town, it drives the selection of patterns from requirements and bottom-up, it supports the traceability of design solutions back to the requirements.

A visualization of the above concepts is given in Figure 2. For ease of reference, in the following of this paper we refer to this as the *Requirements Tree*. In particular, the figure presents the relationship between requirements, objectives, and patterns for the case study of Section 2. Every requirement is connected with a security objective as follows. The analyst chooses the top objective from Figure 1 (integrity, confidentiality,

etc) to which a requirement belongs, and then the relevant sub-objective, if the requirement is fine-grained enough (see R1 and R2 in Figure 2). If the requirement is very high-level and allows the analyst to specify only a top objective, then all sub-objectives should be chosen implicitly (see R3 and R4 in Figure 2). Finally, the patterns which contribute to the satisfaction of the low level objectives are identified. The patterns are shown just to give the complete picture to the reader and will be explained further on in the article.

Example 1 In Figure 2, requirement *Prevent corruption of news items while they are stored in the system (R2)* is connected to *Integrity of stored data*. On the other hand, *Users always must be able to read the news (R3)* is associated to the *availability* top objective. This means that the availability of news stored in the database, its transmission to the customer, and the provisioning of data should be protected against denial-of-service attacks.

4 Metrics for Security Patterns

In our approach, the assessment of a security-enhanced architecture is performed in two stages. First, we must determine the level of protection each group of patterns provides to the architecture. Second, these low level contributions need to be aggregated, using the Requirements Tree, into an overall security indicator. In this section we deal with the first problem and Section 5 describes the second one.

4.1 Mapping threats to objectives

The level of protection of a software system against various classes of threats represents the result of the implementation of security requirements. In order to measure protection against threats, a reference list of threats must be identified. In this work, we use the threats included in Microsoft's STRIDE methodology [6]. This choice is driven by two reasons. First, the threats mentioned in STRIDE are meant to be used for the threat modelling activities performed during the architectural phase of the software development life-cycle. Therefore, they naturally fit into the context of this work. Second, STRIDE is a well-known and mature technique that has been successfully adopted in several software projects. STRIDE might not be exhaustive concerning the range of potential architectural threats. This issue, however, is not in scope for this work.

In STRIDE, threats are organised into tree structures, called *threat trees*. As shown in Figure 4, each tree has a generic threat class as its root (e.g., "DoS against a process") that is decomposed into more specific threats via AND/OR branches. The decomposition can proceed further as necessary, but is limited, on average, to three levels. The resulting trees are similar to the well-known attack trees [15] although STRIDE threats are more general.

Each threat tree can be unambiguously mapped to a corresponding elementary objective of the Objectives Decomposition as shown in Figure 3[2].

4.2 Rating threat severity

Each threat has a different severity degree. For instance, vulnerabilities leading to some threats can be easier to discover. Similarly, carrying out a specific threat scenario can be costlier than others. We assign weights to each elementary threat to capture this difference.

The weights can be determined by a risk assessment method that assigns severities to threats. In this work our preference went naturally to Microsoft's DREAD [12], the companion method of STRIDE. The DREAD approach considers five factors contributing to the severity of a threat:

[2]There is only one minor exception: to match the accountability objectives (i.e., auditability and non-repudiation), the branches of the "repudiation of data flow" threat (i.e., repudiation of transactions and repudiation of messages, respectively) had to be considered.

STRIDE threat trees		Elementary security objectives
Spoofing an external entity or process	$\Longrightarrow$	Authentication
Tampering with data store	$\Longrightarrow$	Integrity of stored data
Tampering with data flow	$\Longrightarrow$	Integrity of transmitted data
Tampering with a process	$\Longrightarrow$	Integrity of application
Repudiate message	$\Longrightarrow$	Non-repudiation
Repudiate transaction	$\Longrightarrow$	Auditability
Information Disclosure of data store	$\Longrightarrow$	Confidentiality of stored data
Information Disclosure of data flow	$\Longrightarrow$	Confidentiality of transmitted data
Information Disclosure of a process	$\Longrightarrow$	Confidentiality of application
DoS against data store	$\Longrightarrow$	Availability of stored data
DoS against data flow	$\Longrightarrow$	Availability of transmitted data
DoS against a process	$\Longrightarrow$	Availability of application
Elevation of Privileges for processes	$\Longrightarrow$	Authorization

Figure 3: Threats to security objectives

- **Damage potential (D):** how great the damage of exploiting the threat could be.
- **Reproducibility (R)**: how easy it is to repeat the attack.
- **Exploitability (E)**: how easy it is to exploit the vulnerability.
- **Affected users (A)**: how many users can be affected if the attack is successful.
- **Discoverability (D)**: how easy it is to discover for an attack that the threat presents in the system.

At this stage, we only consider the DREAD factors that are related to the "likelihood" of the threats, i.e., Reproducibility, Exploitability, and Discoverability—here we do not consider the factors related to the "impact", i.e., Damage potential and Affected Users. The impact is brought into the picture later on in Section 5, where the impact due to the failure of a requirement is considered.

As shown in Figure 4, for each of the factors of interest, an analyst should rate the specific threats (in gray) on a scale from 1 to 3 (as in the DREAD methodology). The severity s_i of a specific (i.e., leaf) threat t_i of a root threat t is then computed as the normalized sum of reproducibility R_i, exploitability E_i and discoverability D_i:

$$s_i = (R_i + E_i + D_i) / \sum_{\forall t_i \in t} (R_i + E_i + D_i)$$

Example 2 *For the* denial-of-service against a process *threat tree (see Figure 4), the following severities can be assigned to the specific threats:* Consume application-specific resources *s=7,* Consume fundamental resource *s=8,* Input validation *s=5,* Tampering with on-disk process *s=4. The total sum equals 24. Dividing each resulting number by 24, the final weights are 0.291, 0.333, 0.210 and 0.166.*

The only exceptions to this method are the threats with AND-decomposition branches. For each AND-branch the sum (coverage) is calculated separately. Then the *minimal sum* among these alternatives is used to normalize all values (for all branches). The rationale behind this is that the success of exploiting the root threat depends on successful compromising of *all* branches. This means that in order to protect the system we need to reduce the probability to compromise at least one branch. In other words, the level of protection of the elementary objective depends on the level of protection of the best protected branch, i.e., the branch which is less likely to be compromised. This allows us to consider only one (i.e., the strongest) branch. This exception is illustrated in the following example.

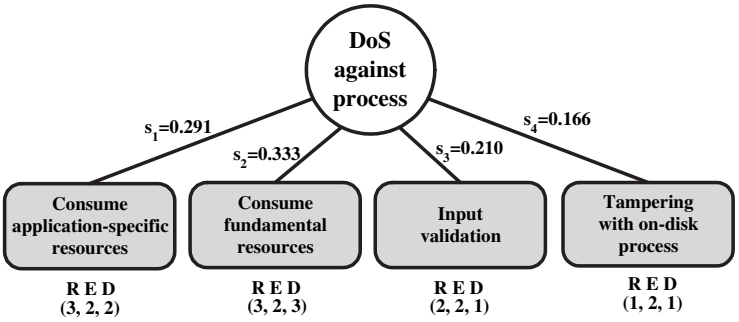


Figure 4: DoS against process tree

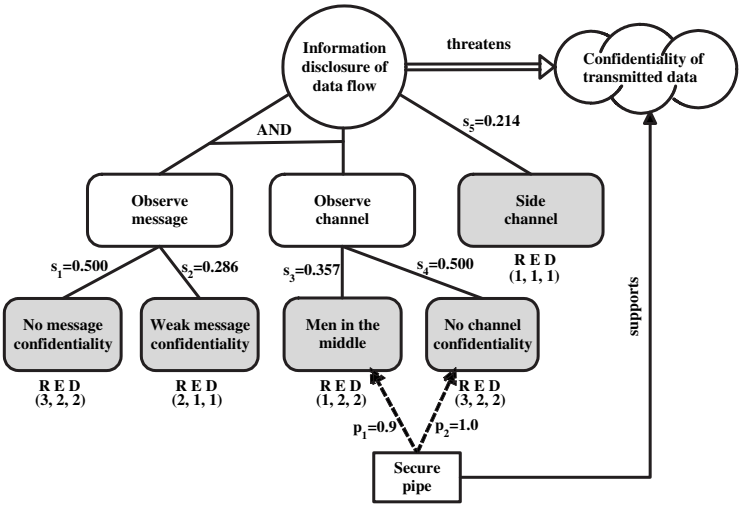


Figure 5: Information Disclosure of data flow tree

Example 3 Consider the threat *Information disclosure of data flow* which violates *Confidentiality of data transmission* in our example (Figure 5). Note that the first two (on the left) and second two elementary threats are AND-branches. That is why the lowest scoring child nodes should be selected when choosing the normalization value and adding the rest values: 7+4=11 vs. 5+7=12. Then adding the score of the another OR branch (i.e., 3) to the minimal sum (i.e., 11) the normalization value becomes: 11+3=14. This results in the following values of 0.500, 0.286, 0.357, 0.500, and 0.213. Note that the sum of all values does not equal 1 (it is greater), since the AND-join implies that only one of the branches should be chosen.

4.3 Determining threat protection

In this section, the security patterns are matched against threats in order to determine the level of protection they offer. This is the building-block metric on which the rest of the approach is based. The level of protection is computed as the prevention of a set of threats associated to the security objectives the pattern contributes to achieve.

Often a pattern provides reasonable protection against a threat but it does not cover it entirely. In some

cases a pattern may offer only marginal protection for a threat. This usually happens when the main contribution of a pattern is to satisfy one objective, and has some side effect on a secondary objective.

The level of protection of a pattern against a threat is determined in a semi-qualitative way, by assigning one of the following values: 0, 0.1, 0.5, 0.9, 1. The idea behind these values is the degree of the attacks (realising a certain threat) that will be prevented if the patterns are applied. These values denote the percentage of threat mitigation, 0 being 'not at all' and 1 being 'completely'. Values 0.1 and 0.9 are preferable to denote side-effects of some patterns and impossibility to satisfy some objectives completely (e.g., availability).

Example 4 *As shown in Figure 5,* SECURE PIPE *provides a 100% protection against* No channel confidentiality *threat of* Information Disclosure of data flow *(1) while cannot give the same certainty for* men-in-the-middle *attack (0.9). Similarly,* REPLICATED SYSTEM *(not shown in the figure) increases the probability to withstand a DoS attacks which are based on* providing invalid inputs *(*DoS against Process*) to a program but does not provide a complete solution (0.5). Finally,* APPLICATION FIREWALL *can provide some side-effect to "access to memory" but this is just a small contribution (0.1).*

In case that several patterns contribute to the same threat, their contributions should be seen as the probability that at least one of the independent protection mechanisms is able to protect the system against this threat, i.e., the 'defense in depth' principle[3]. In other words, if several patterns protect the system against the same leaf threat t_i with coverage $p_{i1}, p_{i2}, ...$ then the overall protection against the threat p_i will be:

$$p_i = 1 - \prod_j (1 - p_{ij})$$

4.4 Computing the overall coverage

Finally, the protection values assigned to leaf threats (via the patterns) can be aggregated to determine the overall coverage for the threat tree. The aggregation can be seen as a weighted function. The computation considers all instantiated patterns supporting a given objective (protecting against corresponding root threat) and produces overall coverage for the objective. The protection values of the patterns (determined according to Section 4.3) are multiplied by the severity of the threats and then summed up:

$$c_t = \sum_{i \in t} p_i * s_i$$

When considering AND-branches, the contributions for each branch are summed up separately, and the difference between the most protected branch (with the lowest sum value) before the implementation of the patterns and after is assigned to the AND-branch. In other words the total coverage of AND branches ($B_1, B_2, ...$) can be calculated as follows:

$$c_{AND} = min(\sum_{i \in B_1}(s_i), \sum_{i \in B_2}(s_i), ...) - min(\sum_{i \in B_1}(s_i * (1 - p_i)), \sum_{i \in B_2}(s_i * (1 - p_i)), ...)$$

The rationale behind this is has been explained in Section 4.2. Note, that after implementation of several patterns another AND branch may become tougher for an attacker.

When the level of protection against the root threat is found, the same number is assigned to the corresponding leaf node at the Requirements Tree.

[3]In general, we acknowledge that the contributions of several patterns to a given objective may not be independent from one another. However, we make this simplification to keep the computation more manageable.

Example 5 For *DoS against a process* threat (see Figure 4), an analyst has defined the following security patters: LOAD BALANCER and REPLICATED SYSTEM. These two security patterns cover *Consume application specific resource, Consume fundamental resource* threats 0.5/0.5 and 0.5/0.5 correspondingly. Weights (as it was determined in Example 2) are 0.291, 0.333, 0.21 and 0.166. The overall protection against *DoS against process* threat is the following sum: $0.291 * (1 - (1 - 0.5) * (1 - 0.5)) + 0.333 * (1 - (1 - 0.5) * (1 - 0.5)) + 0.21 * 0 + 0.166 * 0 = 0.468$.

This part of the analysis is hard for non-experts. In particular, the hardest and error-prone steps are: identifying the contributions of patterns to elementary threats, assigning severity degrees to elementary threats, and identifying benefits of patterns (coverage values). On the other hand, these steps are context independent. This means that the results, once achieved, can be (re-)used in any other system. In other words, the analysis can be done thoroughly by highly experienced security experts and then used for analysis in during system development by (non-expert) security designers. Moreover, pattern providers or a Trusted Third Party organisation can do a number of on-site experiments and get realistic statistics which can be used for more thorough definitions of values. This value must be included in the description of the pattern. We assume that the average protection value of a pattern should be the same in all contexts (e.g., being context-independent).

5 Aggregation of metrics

In order to aggregate the values computed for the elementary objectives, we assign weights to the edges in the Requirements Tree. At the top-most part of the tree, weights are assigned to reflect the contribution of each requirement to the overall security goal (see Figure 2). These weights reflect the impact of each individual requirement, for instance, by quantifying the expected monetary loss if the requirement fails.

Example 6 *Consider the four requirements for the on-line publication system (see Figure 2). The following possible losses can be assigned to each requirement, as follows: R1 = 20,000 $, R2 = 80,000 $, R3 = 100,000 $, R4 = 40,000 $ (total = 240,000 $). Thus the normalised weights will be 0.083, 0.333, 0.417 and 0.167, respectively.*

Weights for other edges can be determined in a similar way. The losses caused by failures of objectives are determined and then normalised for every parent-child relationship in the Requirements Tree.

Given this weighted graph, it is possible to verify how well a set of patterns satisfy the security needs of the system-to-be. First of all, we choose a set of patterns which we would like to test. Their contribution towards satisfying the elementary objectives is determined, as shown in Section 4. Using the weights denoting how the selected security patterns impact the satisfaction of elementary objectives, the contributions of each individual pattern can be aggregated. This is done by applying a weighted function to each objective node, starting from the bottom of the Requirements Tree. This process results in a degree of satisfaction of the general security goal (i.e., the value of the fictitious top node in the Requirements Tree). Algorithm 1, built using our previous work on aggregation of security metrics [11], formally describes the core procedure.

6 Experiment

The experiment mentioned in this section is based on a comparison between two student designs of the system described in Section 2. Two lists of selected patterns from each design were compiled (as shown in Figure 6) and are compared with our methodology.

Algorithm 1 Calculation algorithm

Require: $RT = \langle N, E \rangle$: *Requirements Tree: N - nodes(objectives), E - edges*
SL_{leaf} : *protection values of elementary objectives*;
Ensure: $SL[]$ *Satisfaction value of the all security objectives*
Assign received contribution values to elementary objectives
$SL[N_{leaf}] := SL_{Leaf}$;
For each node store number of incoming edges
$SOURCE[N] := |Incoming(E)|$;
Add elementary objectives to a working set
HEAP-insert(N_{leaf});
while HEAPis not empty **do**
take one node from the working set
HEAP-extract(n'); {randomly}
For each outgoing edge from the node
for $\langle n', n \rangle \in Outgoing(n')$ **do**
Reduce the counter for non-traversed incoming edges
decrement(SOURCE[n]);
if all incoming edges are traversed
if $SOURCE[n] = 0$ **then**
compute weighted function for the new node
for all $n'_i.\langle n'_i, n \rangle \in Incoming(n)$ **do**
$SL[n] = SL[n] + L(\langle n'_i, n \rangle) * SL[n'_i]$;
add the new node to the working set;
HEAP-add(n);

Group 1	Group 2
Secure pipe	Secure pipe
Secure service Facade	Secure service Facade
Session	Session
Secure Logger	Secure Logger
Audit Interceptor	Audit Interceptor
Authentication Enforcer	Container Managed Security
Authorization Enforcer	Check-pointed System
Application Firewall	Comparator-checked Fault-tolerant System
	Replicated System
	Load Balancer

Figure 6: Tested sets of patterns

The teams have chosen two different strategies in selecting the patterns. The first group focuses on selecting a minimal but sufficient set of patterns to achieve the proposed requirements, given only a limited implementation budget. The second team put more effort in providing the best security solution, ignoring implementation cost. To these ends, the first group decided to use an APPLICATION FIREWALL–which contributes to many objectives–when Group 2 decided to use a CHECK-POINTED SYSTEM and COMPARATOR-CHECKED FAULT TOLERANT SYSTEM to increase the integrity of the software, and combine them with a REPLICATED SYSTEM and LOAD BALANCER to raise the protection against DoS attacks.

In Figure 7, the complete Requirements Tree with attached security patterns selected by Group 1 is depicted. The weights attached to the edges represent the importance of the lower objectives to the higher ones. The weights assigned to the edges connecting patterns and elementary objectives have been calculated by the method proposed in Section 4 using STRIDE threat trees. Details are omitted for the sake of simplicity.

Using our approach, the satisfaction of the overall security goal is calculated to be 33,1% for the set of patterns chosen by Group 1. Applying the same strategy to the second set of patterns, the calculated score is

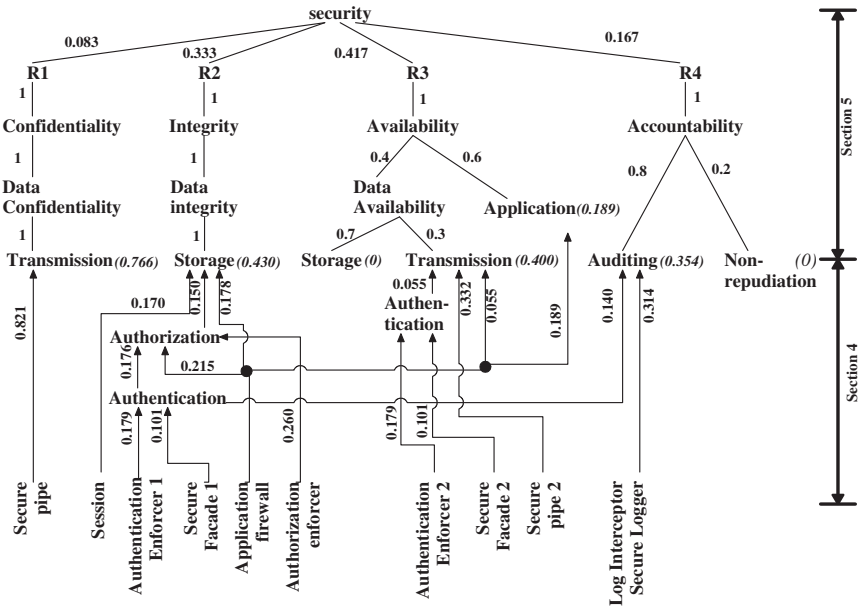


Figure 7: Objectives Decomposition with patterns for the running example.

36,8%. This can be expected, as the second group chose both the REPLICATED SYSTEM and LOAD BALANCER, which contribute greatly to the one of the most important requirement for the publishing system, i.e., *Availability of the application (R3)*. On the other hand, the difference is mitigated somewhat since the APPLICATION FIREWALL has a positive impact on many objectives, including availability. This leads us to the conclusion that, although the second set of patterns provides better protection, the two solutions are comparable in terms of security and the first team designed a more cost-effective solution.

Note that, though the final values are only approximately 35%, this does not indicate that the chosen protection is weak. Even if we assign maximal values to the elementary threats which can be covered on architectural level (i.e., if we assume a 'perfect' architectural solution), the maximum satisfaction level achievable is only 56,8% for this example. Indeed, many threats in STRIDE threat-decomposition trees are only relevant to the detailed design, or even the maintenance phase, of the system. While an argument can be made that the analysis should therefore be conducted without considering design and maintenance threats, the authors are of the opinion that it is important to give a feeling to the software creator about the level of requirements satisfaction at each stage of the software development. A second reason why the computed scores for the examples do not reach 100% quality of protection is that some issues (such as *consume fundamental resources*) cannot be solved perfectly, i.e., there is no such thing as 'perfect security'. A third and final reason is that no patterns are available for many threats which can be solved, in theory, on the architectural level. For example, protection against *Information Disclosure of Data Store* is very poorly represented in the available body of published security patterns.

7 Discussion

The proposed approach applies at the highest level of abstraction of software design in order to identify potential issues as early as possible. In contrast to implemented and deployed software, it is impossible to perform real-world measurements and assess level of protection of the concrete product empirically. For

this reason it is particularly important to reuse context independent measures whose applicability can be extended to several context.

In our approach we used decomposition trees for security objectives and threats. These trees by no means can be proven to be complete, hardly any methodology can provide such guarantee, because the notions of both security and threats are abstract and (most importantly) constantly evolving (e.g., new threats appear). Thus, we have chosen the most complete decompositions that (as of now) suits our purposes.

Further, in the experiments we assigned the contributions and severity levels according to our experience. We must stress that in this paper we propose an approach, but not concrete values. Such values should emerge from either practitioners experience or databases of historical data (which are not available yet).

Finally, further validation is required. However, several challenges are to be faced in this respect. First, in order to get any real-world data one should actually create a software and measure the outcome by comparing several alternatives. Even better, a number of different applications should be tested in order to verify whether the assumptions about context independence really hold. In practice, the software should be exercised in a real setting, which is hard to realise. In alternative, several design variants of a software could be created and evaluated by means of experts. This baseline, then, could be compared with the outcome of the proposed methodology. Another option would be to evaluate existing software application in order to see which one in reality had less design flaws. The problem with this approach is that the design should be based on patterns and the necessary documentation should be. This combination of constraints makes the task of finding suitable case studies particularly tricky (especially in the open source domain).

8 Related work

The problem of security evaluation has received a lot of attention recently. One of the main arguments that makes this problem very important is that the ability to measure security of a system enables choosing the best protection strategy. This leads to the problem of identifying suitable security metrics. Nichols and Peterson [13] proposed and described a number of security metrics which can be used to evaluate the protection against the OWASP top ten vulnerabilities. Although all these metrics are useful for assessment of a specific property, they cannot be used as indicators of the overall security level as they are very specific. Another example of security metric is the attack surface size, which was introduced by Manadhata and Wing [10] as an indicator of the level of security. The method evaluates the functions that the software provides, rather than the security installed to protect the software. Other metrics can be found in [9].

Common Criteria (CC) [7] is a security evaluation method in which a product is evaluated against a specific set of requirements. The product receives an Evaluation Assurance Level (from EAL1 to EAL7) if it satisfies all the requirements for this level. CC is a qualitative method for software evaluation while our approach is based on quantitative assessment though with some subjective values (e.g., level of protection).

CORAS [17] provides a number of ways to model security during software design, and facilitates risk analysis. Risk analysis is more an asset-based methodology, while we focus more on requirements for the software, which allows the application of the approach from the earliest steps of the design.

Also in the area of risk assessment, Clark associates risks to the so-called mission tree, i.e., a tree-based representation of the company specific goals [2]. The evaluation of the risk associated with the mission (i.e., the company top-level goal) is based on the value of the assets at stake.

S. V. Houmb et. al. present a cost-benefit trade-off analysis for aspect-oriented risk-driven development [5]. The idea of the analysis is to use Bayesian Belief Network in order to find the components contributing to Return on Security Investments. Differently from our approach, Houmb proposes a qualitative analysis. Moreover, our main focus is the software base rather than the system and thus we pay more attention to reusable software components (patterns). Also, in our approach we use the same metric (threat coverage) for all nodes of the used tree and thus simplify the analysis.

Finally, the idea of using security patterns to combine security metrics (through security objectives) is also used in [4]. However, the focus of [4] is on assessing and monitoring whether the implemented patterns are working as intended. Our approach evaluates security relative to how well threats are mitigated.

9 Conclusions

In this paper we have presented a novel approach to quantify the security-related qualities of a software architecture adopting security patterns. We use threat coverage as the basic metric for the evaluation. We have also presented an algorithm that aggregates low-level measures associated to these security patterns into an high-level indicator. By applying this algorithm to alternative proposals of software architectures, it is possible to identify the candidate that better satisfies the requirements (in terms of mitigating the threats obstructing the fulfillment of the requirements). The potential of our approach has been illustrated via a case study in the domain of digital publishing.

An interesting research hypothesis is that relationships between patterns (benefit, dependency, impairment, conflict, and alternative) have an impact on the low-level metrics. That is, it is not yet understood how patterns behave, in terms of threat coverage, whenever they are considered in groups. Another interesting issue is the further validation of the approach. These and similar topics are still open to future work.

References

[1] P. Bass et. al. *Software Architecture in Practice*. Addison-Wesley, 2003.

[2] K. Clark et. al. Security risk metrics: Fusing enterprise objectives and vulnerabilities. In *Proc. of IAW*. IEEE, 2005.

[3] P. Clements et. al. *Evaluating software architectures: methods and case studies*. Addison-Wesley, 2002.

[4] T. Heyman et. al. Using security patterns to combine security metrics. In *Proc. of SecSE*. IEEE, 2008.

[5] S. H. Houmb et. al. Cost-benefit trade-off analysis using bbn for aspect-oriented risk-driven development. In *Proc. of ICEECS*. IEEE, 2005.

[6] M. Howard and S. Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.

[7] International Organization for Standardization (ISO/IEC). *Common Criteria for Information Technology Security Evaluation*. Common Criteria Project Sponsoring Organisations, 2.2 edition, January 2004.

[8] ISO/IEC. *Information technology Security techniques Evaluation criteria for IT security*, November 2001.

[9] A. Jaquith. *Security metrics: replacing fear, uncertainty, and doubt*. Addison-Wesley, 2007.

[10] P. Manadhata and J. Wing. Measuring a system's attack surface. Technical Report CMU-TR-04-102, CMU, 2004.

[11] F. Massacci and A. Yautsiukhin. An algorithm for the appraisal of assurance indicators for complex business process. In *Proc. of QoP*. ACM, 2007.

[12] Microsoft. Improving web application security: Threats and countermeasures. available via `http://msdn2.microsoft.com/en-us/library/ms994921.aspx`, June 2003.

[13] E. A. Nichols and G. Peterson. A metrics framework to drive application security improvement. *S&P*, 5(2):88–91, 2007.

[14] R. Scandariato et. al. Architecting software with security patterns. Technical Report CW-515, Katholieke Universiteit Leuven, Department of Computer Science, 2008.

[15] B. Schneier. Attack trees: Modelling security threats. *Dr. Dobb's journal*, December 1999.

[16] SSE-CMM. *Systems Security Engineering Capability Maturity Model - SSE-CMM Model Document*. CMU, version 3.0 edition, June 15 2003.

[17] K. Stolen et. al. Model-based risk assessment - the coras approach. In *Proceedings of the Norsk Informatikkonferanse*, pages 239–249. Tapir, 2002.

[18] K. Yskout et. al. A system of security patterns. Technical Report CW-469, Katholieke Universiteit Leuven, Department of Computer Science, 2006.

Typing Computationally Secure Information Flow in Jif

Liisi Haav[1] Peeter Laud[1,2]
[1]Tartu University [2]Cybernetica AS
{liisi222|peeter.laud}@ut.ee

Abstract

We investigate how to model type systems for computationally secure information flow within the limits of the type system of Jif — an extension of Java with types for tracking the flow of information. In particular, we consider a type system proposed by Laud and Vene which can handle encryption keys as first-class data. We show how the typing decisions of Laud-Vene type system can be captured using the declassification mechanism of Jif, and present a Jif class for "keys" that encapsulates all necessary information releases. The rules that a user of the defined class has to follow, in order to be consistent with the Laud-Vene type system, can be syntactically checked in a straightforward manner.

1 Introduction

The question of *secure information flow* in a program (or a larger system) arises if the program has inputs and outputs of different *security levels*. A common way of specifying secure information flow is *non-interference* [10] stating that inputs of higher security levels must not affect the outputs of lower security levels at all.

There exists various means to statically check whether a program is non-interferent. *Type systems* are one of such means. They are mature enough for being included in software development tools, e.g. Jif [16], an extension of the Java programming language with security levels for values and locations. The Jif compiler statically checks the validity of security annotations, thus ensuring that a program compiles only if it is non-interferent. However, non-interference is often a too strong property for realistic programs. Hence Jif also contains means to *declassify* information; to assign to it a lower security level than would be mandated by the type system.

For programs containing cryptographic operations, non-interference is obviously not the correct formalization for secure information flow. Indeed, a ciphertext depends on the plaintext, and an attacker with an unbounded amount of resources would even be able to find the plaintext from a ciphertext, but nevertheless one would like to consider a program releasing encrypted secrets as secure, because a reasonable attacker cannot deduce anything about the secret inputs from the ciphertexts it sees. A notion of *computational non-interference* is suitable here, demanding only computational, not absolute independence of a program's secret inputs and public outputs. There also exist type systems enforcing such a property for programs containing cryptographic operations; they are more lax than the type systems for non-interference.

The type systems for computational non-interference have not yet found their way to development tools. Some of the simpler systems (e.g. the type system of [19]) can be readily modeled in Jif, using its declassification mechanism to lower the security level of ciphertexts. But these simple type systems put serious restrictions on the manipulation of cryptographic material — they do not consider the encryption keys as the first-class data that can be manipulated by programs. In this paper we investigate how a type system enforcing computationally secure information flow, not putting restrictions on the programs [14], can be modeled in Jif. As a

result of this paper we show that the type system [14] can be modeled in Jif with an exception of encryption cycles. We have to use the declassification mechanism during the encryption, but we also have to track the flow of keys, and consider what happens when information from several sources is combined.

2 Related Work

Static analysis for verifying the programs for secure information flow was started by Denning [8], a suitable type system for a simple imperative language, together with solid semantical underpinnings were first proposed by Volpano et al. [21]. The extensions for the Java type system to track the information flow — the *decentralized label model* — were proposed by Myers [16], subsequently implemented in the Jif compiler. Later, similar extension has appeared for Caml [17]. The enforcement of information flow policies can also be added to a programming language through a suitable library / sublanguage [15]. Among those extensions, Jif is certainly the most mature one, having been used for larger projects, e.g. an e-voting application [6] and a secure e-mail client [11]. A semi-recent overview of language-based information flow security and the static methods to enforce it is given by Sabelfeld and Myers [18].

Language-based analysis of computations containing cryptographic primitives was started by Abadi et al. [2, 1] and extended to a full imperative programming language by Laud [12, 13]. In these papers, a data flow analysis was proposed. A type system similar in expressiveness was proposed by Laud and Vene [14]. Later, several type systems for computationally secure information flow have been proposed. Smith and Alpizar [19] gave a simple type system that handled encryption (but also decryption) with a single key that could not be accessed otherwise. Courant et al. [7] modified this type system to handle deterministic encryption (i.e. pseudorandom permutations). Askarov et al. [3] have proposed an abstraction of the encryption primitive and devised a type system for it. The type system, similarly to [14], does not constrain the operations performed with cryptographic data. Fournet and Rezk [9] give a similar type system to a language containing public-key encryption and signatures and use this language to soundly implement information flow policies for accessing shared memory. Vaughan and Zdancewic [20] have introduced an *information-packing* primitive (which provides confidentiality) and integrated it into the decentralized label model. We are not aware of any previous attempts to model these type systems in Jif.

3 Details of the Jif Type System

Jif is a security-typed language that extends Java programming language by adding types to define different security policies. These policies are expressed by using a decentralized label model. A principal is an entity that can express security requirements.

A principal can also delegate authority to another principal. If principal p delegates authority to principal q then the principal q is said to act for principal p, written $q \succeq p$. Principal $\top$ may act for all other principals whereas principal $\bot$ permits all other principals to act for it. Jif also allows to form principal conjunctions and disjunctions. Principal $p\&q$, a conjunction of principals p and q, allows $p\&q$ to act for both q and p. Principal "p, q", a disjunction of principals p and q, permits both q and p to act for it.

The label consists of two policies: the confidentiality and integrity policy. The confidentiality policy allows the owner to specify which principals may read certain information. It is formed by conjunctions and disjunctions of reader policies $o \to r$, where o is the owner of the policy and r is the specified reader. The policy says that the owner o allows certain information to be read

by principal q only if q is the owner o or if q can act for r. Also, reader policy conjunctions and disjunctions can be formed. The conjunction $c \sqcup d$ enforces both c and d. Thus the information may be read by principal $c \sqcup d$ only if both c and d allow it. The disjunction $c \sqcap d$ says that the information may be read by principal $c \sqcap d$ if one of the two c and d allow it. The least restrictive confidentiality policy is $\bot \rightarrow \bot$, as all principals know that the information may be read by all principals. The most restrictive one is $\top \rightarrow \top$, because only $\top$ can read the information.

The integrity policies are defined dually to confidentiality policies. These policies allow the owner to specify who may have influenced certain information. The writer policy $o \leftarrow w$ says that according to owner o principal q may have influenced the information only if q is the owner o, or may act for w. Similarly to reader policies conjunctions and disjunctions of writer policies can be formed. The most restrictive writer policy is $\bot \leftarrow \bot$, as all principals know that any principal may have influenced the information. The least restrictive writer policy is $\top \leftarrow \top$, because all principals know that only $\top$ may have influenced the information.

Labels are pairs that consist of a confidentiality policy and integrity policy, written $\{c; d\}$, where c is a confidentiality policy and d is an integrity policy. Each variable (whether local, or instance, or class) has a label associated with it; the label of a variable `x` is denoted by `{x}`. These labels must generally be specified by the programmer (otherwise a default label is used), but for local variables, the Jif compiler may be able to infer them automatically. The label of an expression is the join of the labels of its subexpressions; the Jif compiler makes sure that information with a more restrictive label is never stored in a variable with a less restrictive label. To allow such stores, the programmer may *declassify* or *endorse* some expression, relaxing either its associated confidentiality or integrity policy. Whenever some principal's policy is relaxed this way, the Jif compiler verifies that the code currently executing has the *authority* of this principal.

The variables are not the only objects to have labels. Each program point has a label (usually inferred) characterizing the level of the information that has affected the program's control flow whenever it reaches this program point. Only those variables can be assigned to whose label is at least as restrictive as the label of the program point containing the assignment. Jif also relies on explicit label annotations while arguing about interprocedural flow of information. The programmer has to annotate the arguments of methods, as well as their return values. Similarly, all exceptions that a method can throw have labels characterizing the flow of information that lead to the throwing of this exception. To prevent implicit flows through assignments made inside the method, the method also has the *begin-label* that lower-bounds the label of the program points where this method is called from.

The methods in the Jif language may be label-polymorphic. For such methods, the programmer can restrict the polymorphism by declaring that the labels of parameters must satisfy certain constraints. These constraints are verified by the Jif compiler at each program point where this method may be called; these constraints can also be relied on inside the method body. Jif also allows classes to be parameterized by labels or principals; this mechanism is similar to the generics of Java. Each time an object is created, these label and principal parameters are instantiated.

The above description of Jif covers only the features our implementation uses. For a complete overview we refer to [5].

4 Laud-Vene Type System

The type system [14] has been designed to check for the computational security of information flow (CSIF) in a simple imperative language (the WHILE-language) where the set of operations has been extended with key generation (nullary) and encryption (binary). In defining CSIF, let

us assume that each variable of the program has either *high* or *low* security level. A program has CSIF if the initial values of its high-security variables are *computationally independent* of the final values of its low-security variables. Two random variables $\mathbf{X}$ and $\mathbf{Y}$ are computationally independent if no efficient algorithm, given the values of $\mathbf{X}$ and $\mathbf{Y}$ can tell whether these values come from the same experiment (a run of the program) or different experiments.

A *typing* assigns a type to each variable. A *type system* puts restrictions on possible typings. These restrictions make sure that the information does not flow from high-typed (or -security) variables to low-typed variables, unless it is encrypted inbetween. The set of types is quite rich, because there are various kinds of secrets whose flow has to be kept track of. Besides the secret inputs we also have to track the various keys. Let $\mathcal{G}$ be the set of program points where keys are generated. Let the set of *basic secrets* be $\mathcal{T}_0 = \{\mathsf{h}\} \cup \mathcal{G}$ where h denotes the secret inputs. The elements of $\mathcal{T}_0$ denote the various kinds of data that need protection. The set of *encrypted secrets* is $\mathcal{T}_1 = \{\{t\}_N \,|\, t \in \mathcal{T}_0, N \subseteq \mathcal{G}\}$. The type $\{t\}_N$ denotes the amount of information contained in a basic secret t that has been protected (through encryption) by at least one key from each of the key generation statements in N. The encrypted secrets are ordered by the amount of information still present in them: we have $\{t\}_N \leq \{t'\}_{N'}$ if $t = t'$ and $N \supseteq N'$. An *information type* of a variable is basically a set of encrypted secrets, meaning that the value of the variable can depend from only these encrypted secrets (the information type of a variable depending only on public data is the empty set). Given an information type $T \subseteq \mathcal{T}_1$, it may be possible to *simplify* it. First, we may drop from T all its non-maximal elements. Second, if $\{t\}_{N\cup\{i\}} \in T$ and $i \in T$ (here i denotes $\{i\}_\emptyset$) for some $i \in \mathcal{G}$ then we may replace $\{t\}_{N\cup\{i\}}$ with $\{t\}_N$. This simplification corresponds to the ability to decrypt with known keys.

Besides the information type, each variable also has the *usage type*. It is either Key_N, denoting that the variable can be used as a key and its value was generated in one of the program points in $N \subseteq \mathcal{G}$, or it is Data, denoting that the variable is not a key.

Let γ be a typing; it assigns a pair $\langle T, U\rangle$ of information and usage types to each variable. For some variable x, let $\gamma_{\mathsf{Data}}(x)$ be its information type if it is considered to be non-key: if $\gamma(x) = \langle T, \mathsf{Data}\rangle$ then $\gamma_{\mathsf{Data}}(x) = T$, and if $\gamma(x) = \langle T, \mathsf{Key}_N\rangle$ then $\gamma_{\mathsf{Data}}(x) = T \cup N$ (possibly simplified). Each statement $x := o(x_1, \ldots, x_k)$ in the program imposes certain constraints on γ. For a "non-special" o we just require that $\pi_2(\gamma(x)) = \mathsf{Data}$, $\pi_1(\gamma(x)) \geq \gamma_{\mathsf{Data}}(x_i)$ and $\pi_1(\gamma(x)) \geq T_{\mathsf{pc}}$ where T_{pc} is the current *program counter label* (the least upper bound of all $\gamma_{\mathsf{Data}}(b)$, such the execution of this program point is controlled by an *if*- or *while*-statement whose guard is b).

For a key generation statement $x := \mathcal{G}en()$ at the program point i we have the constraints $\pi_2(x) = \mathsf{Key}_N$ for some $N \supseteq \{i\}$ and $\pi_1(x) \geq T_{\mathsf{pc}}$. An assignment $x := y$ can be typed as in the previous paragraph, but if $\pi_2(\gamma(y)) = \mathsf{Key}_N$ then we may also choose to satisfy the constraints $\pi_2(\gamma(x)) = \mathsf{Key}_{N'}$ where $N \subseteq N'$, $\pi_1(\gamma(x)) \geq \pi_1(\gamma(y))$ and $\pi_1(\gamma(x)) \geq T_{\mathsf{pc}}$. Finally, for a statement $x := \mathcal{E}nc(k, y)$, where $\pi_2(\gamma(k)) = \mathsf{Key}_N$ we may also choose to satisfy the constraints $\pi_2(\gamma(x)) = \mathsf{Data}$, $\pi_1(\gamma(x)) \geq T_{\mathsf{pc}}$ and $\pi_1(\gamma(x)) \geq \{\{t\}_{M\cup\{i\}} \,|\, \{t\}_M \in \pi_1(\gamma(k)) \cup \pi_1(\gamma(y)), i \in N\}$.

As we mentioned before, the type of a secret input variable must be at least $\langle\{\mathsf{h}\}, \mathsf{Data}\rangle$. Similarly, the *least upper bound* of $\gamma_{\mathsf{Data}}(y)$ for all public outputs y must not be $\{\mathsf{h}\}$ or higher. If these conditions and all constraints from previous two paragraphs hold, then the program has computationally secure information flow. Note, however, that the security of the encryption scheme under *key-dependent messages* [4] is necessary for this to hold. In [14] the simplification rules for sets of encrypted secrets were more complex, allowing the type system to detect *encryption cycles* and deem them insecure. These rules are probably too complex to be modeled within the Jif type system. Still, even without considering encryption cycles, the modeling task remains interesting.

5 The Principles of Modeling

There is a special `Key` class defined so that only keys generated by that class can be used for encryption. The class implements the methods for generating new keys (realized in the constructor) and encryption. As we also want to allow one to access the "actual value" (as a bit-string) of the key, there will also be a method that returns it.

In [14], a type consisted of an information type and a usage type. We will obviously use Jif's labels to track the information types of values. A value has the usage type Key_N only if it is an object of class `Key`, otherwise it has the usage type `Data`. The class `Key` is parameterized with something that allows us to track the set N of possible generation points of that key.

The public and private inputs are modeled by having a fixed principal H that is allowed to read private data (the public data can be read by $\bot$). For each key generation point $g \in \mathcal{G}$ we also introduce a principal P that is allowed to read the keys generated at this point. Besides P, we also introduce the principal $\overline{P}$ that certainly *does not know* the keys generated at the point g. On can form conjunctions and disjunctions of these principals. For example, $P_1 \& \overline{P_2}$ is a principal that knows the keys generated at g_1, but certainly does not know the keys generated at g_2. The principal $P \& \overline{P}$ does not exist — i.e. it is considered to be equivalent to $\top$ and it may not occur in the program text.

If information in variable x can be read by someone who acts for the principal X and the key k was generated at g (represented by the principals P and $\overline{P}$), then the ciphertext $\mathcal{E}nc(k, x)$ may be read by someone who acts for the disjunction $X, \overline{P}$. The necessary declassification from X to $X, \overline{P}$ is performed by the encryption method of the `Key`-class. Similarly, if a key k_1 generated at g_1 is encrypted with a key k_2 generated at g_2 then the result can be read by $P_1, \overline{P_2}$. A pair consisting of $\mathcal{E}nc(k_2, k_1)$ and $\mathcal{E}nc(k_3, k_2)$ (where k_3 is generated at g_3) can be read by the principal $((P_1, \overline{P_2}) \& (P_2, \overline{P_3}))$. This means that the result may be read by either a principal who acts for $P_1 \& P_2$ (the principal who may already read both plaintexts), or principal who acts for $\overline{P_2} \& \overline{P_3}$ (the principal who surely does not know the keys to decrypt both ciphertexts), or principal who acts for $P_1 \& \overline{P_3}$ (the principal who knows the key k_1 but is unable to decrypt $\mathcal{E}nc(k_3, k_2)$. The fourth possibility $P_2 \& \overline{P_2}$ equals $\top$.

In [14], a program has computationally secure information flow if the least upper bound of the types of its public variables is not `h` or greater. While modeling this type system in Jif, the programmer has to explicitly state that least upper bound. The public variables may be read by a principal of the form $P_{i_1} \& \ldots \& P_{i_k} \& \overline{P_{j_1}} \& \ldots \& \overline{P_{j_l}}$ with $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_l\} = \emptyset$. The use of this label is more clearly explained in Sec. 7.

6 The `Key`-class

The class for encryption keys, given in Fig. 1, contains methods for key generation, encryption and for returning the value of the key. In the declaration of a variable that is of type `Key`, written `Key[l1,l2]{l} k`, the label `l1` must be of the form `{p->P1&...&Pn;p<-*}` and `l2` of the form `{p->NotP1&...&NotPn;p<-*}` for some principal `p` that has the authority to execute the declaration of `k`. In the syntax of Jif, `*` denotes $\top$. Jif does not check that the two labels are of the correct form (first one containing $P_1 \& \cdots \& P_n$ and the second one $\overline{P_1} \& \cdots \& \overline{P_n}$), but this syntactic check could be easily included somewhere in the development environment. The variable `k` may contain keys generated at one of the points $g_1, \ldots, g_n$. The covariance of the label parameters is used in the subtyping decisions; this allows the assignments of the form `k1 = k2;` where the keys pointed to by `k1` may have been created in at least as many program points as the keys pointed to by `k2`.

```
1  import value.Value;
2  import javax.crypto.*;
3  import javax.crypto.spec.*;
4
5  public class Key[covariant label l1, covariant label l2] {
6    final byte[]{this} key;
7
8    Key() {
9      this.key = gen();
10   }
11
12   String{pt meet l2 ; p<-*}
13     encrypt{this}(principal p, String pt)
14       where {pt}<={p->*;p<-*},{this}<={p->*;p<-*},caller(p)
15   {
16     String r = encAES(key,pt);
17     return declassify(r, {pt meet l2 ; p<-*});
18   }
19
20   String{this ; l1} value() {
21     String{this ; l1} keyValue = Value.bytesToString(key);
22     return keyValue;
23   }
24
25   private static byte[] gen() {  ...  }
26
27   private static String encAES(byte[] raw,String pt) {  ...  }
28 }
```

Figure 1: The class `Key`

The instance method `encrypt` takes a principal `p` and plaintext `pt` as arguments. The principal `p` must have authorized the call to `encrypt`, as stated by the precondition `caller(p)`. This is needed because the method uses declassification to properly model the weakening of the restrictions on information flow according to type system [14] and thus the authority of the concerned principal is required. The other two restrictions `{pt}<={p->*;p<-*}` and `{this}<={p->*;p<-*}` assure that labels `{pt}` and `{this}` for the plaintext and the key only contain policies of principal `p`. See Sec. 9 for a discussion of this restriction. As the actual encryption method `encAES` uses both `key` and `pt` then the label of the result `r` is a conjunction of the two labels: `{this;pt}`. The label of the ciphertext `r` is then declassified to also allow the text to be read by the principal who surely does not know the key used for encryption. Thus the new label of the ciphertext is a disjunction `pt meet l2`. As the label `{r}` only contains the policy of the principal `p`, there exists sufficient authority to perform the declassification.

The `Key`-class also contains the method `value` for returning the actual value of the key. The method will just convert the array of bytes `key` to a string and restrict its label. This restriction is manifested as the label of the return value of the `value`-method.

If a program makes use of the class `Key` and does not contain any declassification statements

```
1  public static final void main{p<-*}(principal{p<-*} p, String args[])
2      throws IllegalArgumentException
3        where caller(p)
4  {
5
6    final label L = new label {p->NotP1; p<-*};
7    PrintStream[{*L}] out = ...
8
9    Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k;
10   k = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();
11
12   String{p->H;p<-*} pt = "Plaintext";
13   String x = k.encrypt(p, pt);
14   out.println("x: " + x);
15 }
```

Figure 2: Example 1

outside of this class, then Jif's type system puts "the same" restrictions on it as the type system of Laud and Vene. We believe that this statement could be formalized as a theorem; however we do not currently intend to do so. It would require a formalization of the Jif type system to an extent that we are not aware of having been done. Also, our goal has been "similarity" of the behavior of type systems, not their total coincidence. Indeed, we have already seen a difference (the encryption cycles) between the two type systems. The next section shows that there are other differences as well.

7 Examples

Fig. 2 presents the most basic example of using the `Key`-class. We generate a key (the two principals associated with this key generation statement are `P1` and `NotP1`), use it to encrypt a secret (denoted by having the confidentiality policy `p->H`) plaintext and output a ciphertext. The program is secure according to the type system of [14] and the Jif compiler also accepts it.

Similarly to Java, Jif starts the execution of the program from a method named `main`, with the correct signature. In Jif's case, this signature also includes the principal `p`, under whose authority the program is executed. Jif's standard library includes the labeled versions of input and output streams; in our example, `out` is an output stream that can print values whose labels are no more restrictive than `L`. By defining the label `L`, we are stating the least upper bound of the labels of the public variables, as required in the end of Sec. 5. Jif does not verify that `L` is of the shape required in Sec. 5, but this simple syntactic check could be embedded elsewhere.

But if in addition to ciphertext `x` the value of the key `k` is also output (as shown in Fig. 3, where the method signature is no longer shown), then Jif rejects the program because the label of `k.value()` contains the policy `{p->P1}` which is not less or equal to `L`. We cannot add this policy to `L` (although the Jif compiler would not complain) because that would violate the conditions put on `L` in Sec. 5.

In the example in Fig. 4 two keys `k1` and `k2` are defined. The first is used to encrypt the plaintext, while the second is used to encrypt the first key. Both ciphertexts are output. The Jif compiler accepts this program, because the first ciphertext may be read by `NotP1` (and also

```
{
  final label L = new label {p->NotP1; p<-*};
  PrintStream[{*L}] out = ...

  Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k;
  k = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();

  String{p->H;p<-*} pt = "Plaintext";
  String x = k.encrypt(p, pt);
  out.println("x: " + x);
  out.println("k: " + k.value());
}
```

Figure 3: Example 2

```
{
  final label L = new label {p->NotP1&NotP2; p<-*};
  PrintStream[{*L}] out = ...

  Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k1;
  k1 = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();

  Key[{p->P2;p<-*}, {p->NotP2;p<-*}] k2;
  k2 = new Key[{p->P2;p<-*}, {p->NotP2;p<-*}]();

  String{p->H;p<-*} pt = "Plaintext";

  String x1 = k1.encrypt(p, pt);
  String x2 = k2.encrypt(p, k1.value());

  out.println("x1: " + x1 + " x2: " + x2);
}
```

Figure 4: Example 3

by `H`), while the second ciphertext may be read by `NotP2` (and also by `P1`). The label `L` is of the correct form.

Fig. 5 demonstrates double encryption: here `x2` is a ciphertext that may be read by each of the principals `H`, `NotP1` and `NotP2`. Hence we may output `x2` and also one of the keys (chosen statically). Note the value of the label `L` — it states that the key(s) generated at the program point g_1 are public.

The key that is used for encryption may also depend on other values as shown in figure 6. The value of the key `k3` depends on some other (public) value. The labels `l1` and `l2` in the declaration of `k3` have to contain both `P1`/`NotP1` and `P2`/`NotP2`, otherwise the assignment to `k3` is not allowed because of incompatible types. After encrypting the plaintext `pt` with the key `k3` the ciphertext `x` may be read by principal `NotP1&NotP2` (but not just `NotP1` or `NotP2`). Still,

```
{
  final label L = new label {p->P1&NotP2; p<-*};
  PrintStream[{*L}] out = ...

  Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k1;
  k1 = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();

  Key[{p->P2;p<-*}, {p->NotP2;p<-*}] k2;
  k2 = new Key[{p->P2;p<-*}, {p->NotP2;p<-*}]();

  String{p->H;p<-*} pt = "Plaintext";

  String x1 = k1.encrypt(p, pt);
  String x2 = k2.encrypt(p, x1);

  out.println("x2: " + x2 + " k1: " + k1.value());
}
```

Figure 5: Example 4

```
{
  final label L = new label {p->NotP1&NotP2; p<-*};
  PrintStream[{*L}] out = ...

  Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k1;
  k1 = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();

  Key[{p->P2;p<-*}, {p->NotP2;p<-*}] k2;
  k2 = new Key[{p->P2;p<-*}, {p->NotP2;p<-*}]();

  Key[{p->P1&P2;p<-*}, {p->NotP1&NotP2;p<-*}] k3;
  k3 = (..) ? k1 : k2;

  String{p->H;p<-*} pt = "Plaintext";
  String x = k3.encrypt(p, pt);
  out.println("x: " + x);
}
```

Figure 6: Example 5

the label `L` allows us to output `x`.

The example in Fig. 7 differs from the previous one only in the confidentiality policy on the information from which there is an implicit flow to `k3`. That policy is now `{p->P3}`, instead of `{}`, and that is also the confidentiality policy of `k3` itself. According to the type system of [14], the ciphertext `x` can now be read by either of the principals `H&P3` and `NotP1&NotP2`. Hence the program is still secure and it should suffice to define `L` as in Fig. 7.

```
{
  final label L = new label {p->NotP1&NotP2; p<-*};
  PrintStream[{*L}] out = ...

  Key[{p->P1;p<-*}, {p->NotP1;p<-*}] k1;
  k1 = new Key[{p->P1;p<-*}, {p->NotP1;p<-*}]();

  Key[{p->P2;p<-*}, {p->NotP2;p<-*}] k2;
  k2 = new Key[{p->P2;p<-*}, {p->NotP2;p<-*}]();

  Key[{p->P3;p<-*}, {p->NotP3;p<-*}] k;
  k = new Key[{p->P3;p<-*}, {p->NotP3;p<-*}]();

  Key[{p->P1&P2;p<-*}, {p->NotP1&NotP2;p<-*}] k3;
  k3 = (k.value()) ? k1 : k2;

  String{p->H;p<-*} pt = "Plaintext";
  String x = k3.encrypt(p, pt);
  out.println("x: " + x);
}
```

Figure 7: Example 6

Jif, however, acts differently. It considers there to be information flow from the target of the method call (`k3`) to the result of the method call (`x`). Hence, according to Jif, `x` can be read by either of the principals `H&P3` or `NotP1&NotP2&P3`. The program in Fig. 7 does not compile. We could add `P3` to `L`, thereby making it compile again, but then we could not output any ciphertexts created with the key `k` (this is still allowed in [14]).

Such an assumption by Jif cannot probably be overcome, as long as `encrypt` is an instance method and the occurrences of declassification are constrained to be inside the class `Key`. On the other hand, this assumption is also not a weakness of Jif, because it is necessary each time the target of the call is `nil`. The language used in [14] does not allow the possibility of a key being undefined.

8 Static Method for Encryption

The assumption made by Jif on the information flow from that target of a method call to the result of that call could be overcome if we implement encryption as a static method, as shown in Fig. 8. In addition to a principal `p` and a plaintext `pt`, the static encryption method takes a key `k` as an argument. The method saves `k.key` in `kv`, but before dereferencing `k` it declassifies it, such that the possible `NullPointerException` that may be thrown does not have a too restrictive label. After saving `k.key` in `kv`, the static method works the same way as the instance method.

This static encryption method is typed almost like in the type system [14] with the exception of handling the `NullPointerException`. Using this method instead of the instance method would allow us to compile the example in Fig. 7. Besides the fact that handling the `NullPointerException` is not correct according to the type system [14], handling it also might

```
static String{pt meet l2; k meet l2; p<-*}
  encrypt_s{p<-*}(principal p,Key[l1,l2] k,String pt)
    :{pt meet l2; k meet l2; p<-*}
      throws NullPointerException
        where {pt} <= {p->*;p<-*}, {k} <= {p->*;p<-*},
          caller(p)
{
  byte [] kv = declassify(k,{k meet l2; p<-*}).key;
  String r = encAES(kv,pt);
  return declassify(r, {pt meet l2 ;k meet l2;p<-*});
}
```

Figure 8: Static encryption method

be cumbersome and thus the instance encryption method is used where implicit information flow does not occur.

9 Conclusions and Discussion

We have shown that existing tools for secure programming (in particular, Jif) are also well-suited for making sure that programs have computationally secure information flow. The next logical step would be the extension of Jif to include encryption in its policies. So far we have not really made use of the decentralized label model; in fact, we have actively tried to work around it by stating that there is a single principal `p` whose policies we are concerned with. Indeed, Laud and Vene [14] also do not consider multiple principals. Hence the extensions of this work also have to consider important theoretical problems, e.g. the integrity of keys.

References

[1] Martín Abadi and Jan Jürjens. Formal Eavesdropping and Its Computational Interpretation. In Naoki Kobayashi and Benjamin C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001*, volume 2215 of *LNCS*, pages 82–94, Sendai, Japan, October 2001. Springer-Verlag.

[2] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *International Conference IFIP TCS 2000*, volume 1872 of *LNCS*, pages 3–22, Sendai, Japan, August 2000. Springer-Verlag.

[3] Aslan Askarov, Daniel Hedin, and Andrei Sabelfeld. Cryptographically-Masked flows. In Kwangkeun Yi, editor, *SAS*, volume 4134 of *Lecture Notes in Computer Science*, pages 353–369. Springer, 2006.

[4] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.

[5] Stephen Chong, Andrew C. Myers, K. Vikram, and Lantian Zheng. *Jif Reference Manual*, April 2008.

[6] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society, 2008.

[7] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2007.

[8] Dorothy E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.

[9] Cédric Fournet and Tamara Rezk. Cryptographically Sound Implementations for Typed Information-Flow Security. In *POPL 2008, Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Francisco, California, January 2008. ACM Press.

[10] Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, California, April 1982. IEEE Computer Society Press.

[11] Boniface Hicks, Kiyan Ahmadizadeh, and Patrick Drew McDaniel. From languages to systems: Understanding practical application development in security-typed languages. In *ACSAC*, pages 153–164. IEEE Computer Society, 2006.

[12] Peeter Laud. Semantics and Program Analysis of Computationally Secure Information Flow. In David Sands, editor, *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001*, volume 2028 of *LNCS*, pages 77–91, Genova, Italy, April 2001. Springer-Verlag.

[13] Peeter Laud. Handling Encryption in Analyses for Secure Information Flow. In Pierpaolo Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003*, volume 2618 of *LNCS*, pages 159–173, Warsaw, Poland, April 2003. Springer-Verlag.

[14] Peeter Laud and Varmo Vene. A Type System for Computationally Secure Information Flow. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *15th International Symposium on Fundamentals of Computation Theory (FCT) 2005*, volume 3623 of *LNCS*, pages 365–377, Lübeck, Germany, August 2005. Springer-Verlag.

[15] Peng Li and Steve Zdancewic. Encoding information flow in haskell. In *CSFW*, page 16. IEEE Computer Society, 2006.

[16] Andrew C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, Texas, January 1999. ACM Press.

[17] François Pottier and Vincent Simonet. Information flow inference for ml. *ACM Trans. Program. Lang. Syst.*, 25(1):117–158, 2003.

[18] Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.

[19] Geoffrey Smith and Rafael Alpízar. Secure Information Flow with Random Assignment and Encryption. In *4th ACM Workshop on Formal Methods in Security Engineering*, pages 33–43, 2006.

[20] Jeffrey A. Vaughan and Steve Zdancewic. A cryptographic decentralized label model. In Birgit Pfitzmann and Patrick McDaniel, editors, *IEEE Symposium on Security and Privacy*, pages 192–206. IEEE Computer Society, 2007.

[21] Dennis M. Volpano, Geoffrey Smith, and Cynthia Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.

Persona-based Identity Management: A Novel Approach to Privacy Protection

Mohammed Hussain and David B. Skillicorn
School of Computing, Queen's University
Kingston, ON, Canada K7L 3N6
{*hussain,skill*}@*cs.queensu.ca*

Abstract

Protecting privacy is often conceived and implemented as a matter of controlling the *flow* of identity and identity-related information. This, for instance, is the strategy of much privacy legislation and business privacy policy. Such approaches are increasingly weakened by the ability of data-mining techniques to fuse these *partial identities* into a complete identity, even using attributes that are not normally considered to be identifying.

We propose instead that the solution to protecting privacy is the use of *artificial identities*, generated robustly using cryptographic techniques and managed by individuals as they wish (with ultimate traceability when required by, for example, law enforcement). The existence of a framework that can protect privacy even in the presence of large-scale data collection and analysis makes it possible for individuals and their agents to participate in online activity without the reluctance that is increasingly associated with current approaches.

1 Introduction

The overwhelming benefits of conducting business online, for organizations as well as individuals, have led to a fast-paced growth of web services and applications, such as e-commerce, e-banking, e-universities, and e-government. Managing individuals' identities, attributes, preferences, and privileges is an important issue. Systems that facilitate the management of such information are called identity-management systems (IMS).

A feature expected of IMS is preserving individual privacy. The fear of loss of privacy or identity theft may limit willingness to use web services. A recent survey by the Center for the Digital Future at the University of Southern California showed that more than 60 percent of Americans are extremely worried about their privacy when shopping online [6]. This represents a significant increase from the 47 percent recorded in 2006. Many identity-management systems, both commercial and research, have been developed or extended to handle privacy better. Some well-known examples are Liberty Alliance [14], Shibboleth [20], WS-Federation [21], CardSpace [8], U-Prove [10], Idemix [3], and Prime [2].

Although these systems differ in their approaches to privacy, they agree on assumptions and strategies. These systems work with partial identities and regulate the flow of data involving these identities. They share the assumption that privacy is achieved by allowing only "safe" flows. Several issues make this assumption and strategies based on it inadequate to protect individual privacy.

1. Knowledge-discovery and data-fusion technologies are able to associate partial identities and so to allow reconstruction of good approximations to complete identities. This is true even if each partial identity does not contain data that would normally be considered as identifying. For example,

[1]This work has been partially funded by the Privacy Commissioner of Canada

the work of [11, 17] show that anonymized users' records in movie rating datasets can be traced back to the actual users with the help of a little auxiliary information (*e.g.*, a user mentioning the name of a movie in a public forum). Another example appears in [15], where the genomic data of anonymized patient records are linked to the actual patients.

2. Individuals may not be able to see the boundaries between organizations with whom they share information. For example, they may deal with two apparently-different organizations that are actually part of the same conglomerate and which may feel entitled to share information internally.

3. Federated IMS, such as Liberty Alliance and WS-Federation, regulate the flow of partial identities between identity providers and service providers using privacy policies and individual consent. Ill-designed policies and/or consents without full awareness may lead to catastrophic violations of individual privacy.

4. Federated IMS require identity providers to be available whenever individuals interact with service providers. This in itself may give enough information for identity providers to profile individuals.

5. Anonymous credential systems, such as U-Prove and Prime, use zero-knowledge proof of knowledge techniques instead of policies and consents. This removes the necessity for policies and results in anonymous transactions; however, accountability is maintained by giving a trusted third party the power to de-anonymize individuals. This contradicts the motivation of using anonymous certificate systems in the first place. Moreover, unnecessary replicas of an individual's identity are created every time that individual is involved in a transaction.

These issues suggest that protecting individual privacy requires reconsidering conventional assumptions and strategies.

1.1 A novel approach to privacy

The underlying problem with the previously-mentioned IMS is that individuals' real identities are involved in individuals' transactions, if only in a limited way. To overcome this problem we suggest the use of *artificial identities*, called personas. Personas do not store partial identities. Instead, a persona is just a way to ensure SPs that there is an IDP acting as a guarantor for the individual possessing that persona. This implies that it is much more difficult, if not impossible, to fuse personas. We can think of a persona as acting as an identity representing someone in one or more interactions, for example, a short-term email address that someone uses to subscribe to a service, or one-time use credit card. The goals of the proposed approach are:

i. Empowering individuals to interact with service providers through personas, eradicating the risk of exposing real identities.

ii. Enabling individuals to create their desired personas and limiting the role of identity providers in certification.

iii. Allowing service providers to verify individuals' claims and to determine whether personas are presented by their rightful owners, both without the need to contact identity providers.

iv. Giving identity providers the means to trace personas to the individuals they represent when this is required by a legal process.

These goals allow an individual to reap the benefits of enhanced service at organizations by being a good customer, without allowing these organizations to realize that these good customers are all the

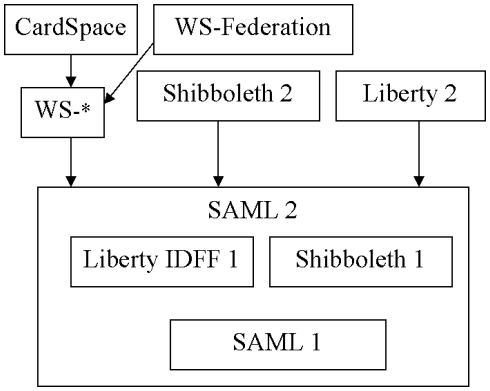


Figure 1: The relation among the various IMS

same person. Thus the individual controls the sharing of information among organizations, rather than the organizations controlling it, perhaps without knowledge or consent.

The contribution of this paper is a novel approach to privacy that achieves these goals, and an architecture and design of a persona-based IMS that supports the laws of identity given in [5].

Section 2 describes various active commercial and research IMS. Section 3 draws the line between persona-based IMS and other IMS. Our approach to privacy protection and an IMS based on that approach are described in detail in Section 3. Section 4 addresses the design of one instance of the presented IMS. In Section 5, a discussion is provided and some open problems are described.

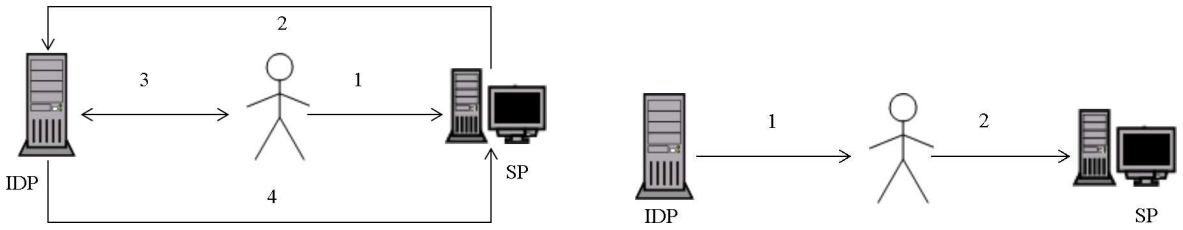
2 Identity Management

The two areas closely related to this proposed design are federated identity management and anonymous-certificate systems. Liberty Alliance, Shibboleth, WS-Federation, and CardSpace are examples of federated IMS. U-Prove and Prime are anonymous-certificate systems. Liberty Alliance and Shibboleth are based on the Security Assertion Markup Language (SAML) [18]. WS-Federation and CardSpace are based on the Web Services (WS-*) framework.

2.1 Liberty Alliance and Shibboleth

Liberty Alliance is a consortium that includes Sun, HP, General Motors, and many other global corporations. The consortium's mission is to provide open standards for the development of federated identity-management systems. The architecture advocated by Liberty Alliance has three frameworks: the Identity Federation Framework (IDFF), the Identity Web-Services Framework (IDWSF), and the Identity Services Interface Specifications (IDSIS). IDFF provides a single means for sign-on for individuals, and simple session management. IDWSF uses IDFF to establish permission-based attribute sharing, service discovery and description, and the associated security profiles. IDSIS builds on both IDFF and IDWSF to support a wide range of applications such as e-wallets and calendar services [16]. Shibboleth from Internet 2 is an open-source IMS for single sign-on across organizations. Shibboleth is used mainly by educational institutions.

Liberty Alliance and Shibboleth use SAML for specifying the communication of identity-related information between individuals and providers. SAML 2 extends SAML 1.1 to include the Liberty Alliance IDFF and Shibboleth. Therefore, Liberty Alliance 2 and Shibboleth 2 use SAML 2 as a basis for secure communication and for basic identity services such as single sign-on [16].



(a) Federated identity management systems (b) Anonymous certificate systems

Figure 2: Typical scenarios in IMS

2.2 WS-Federation and CardSpace

Founded by Microsoft and IBM, WS-federation is a component in the WS-* framework. As with other federated IMS, WS-federation defines how identities can be federated among different providers. WS-federation shares the same goals as Liberty Alliance, but differs in the design and technologies used. CardSpace from Microsoft is an IMS focusing on the individual perspective. Identities are managed at individuals' machines. CardSpace is also based on WS-*. Figure 1 shows the relation among the various IMS.

A typical usage scenario in Liberty Alliance, Shibboleth, WS-Federation and CardSpace goes as follows (refer to Figure 2a). An individual visits a service provider's (SP) site. To authenticate this individual, the SP redirects the individual to her identity provider's (IDP) site. The IDP authenticates the individual and redirects her to the SP, where the redirection message contains a proof that the individual has been authenticated.

2.3 U-Prove, Idemix, and Prime

U-Prove from Credentica, recently acquired by Microsoft, is an anonymous-certificate system. U-Prove focuses on security and privacy aspects of identity management. Prime is another anonymous-certificate system funded by the European Union and several corporations. Similar to U-Prove is Idemix, developed at IBM. In addition to privacy, Idemix also tackles the problem of sharing credentials. Prime attempts to illustrate how European regulations regarding privacy can be incorporated into IMS. U-Prove, Idemix, and Prime help individuals controlling the flow of their identity-related information.

A typical usage scenario in U-Prove, Idemix Prime goes as follows (refer to Figure 2b). An individual visits an SP site. To authenticate to the SP, the individual uses a certificate obtained previously from her IDP. The individual uses zero-knowledge proofs to convince the SP that the individual has a valid certificate from the IDP.

3 Persona-based Identity Management

The idea of artificial identities has already been used in a small way. Many people use throwaway email addresses as identifiers to sites that require them, since it is easy to disassociate oneself from such an address if it is misused by the site owner. Cash is a form of anonymous, but guaranteed, financial identity. Historically, letters of introduction served the same purpose. Our approach is an attempt to formalize this behavior by specifying the required requirements, architecture, and operations.

This section presents the building blocks needed to realize the proposed persona-based IMS. First, we start by defining some terminology.

Persona: A persona is an artificial identity that represents an individual in one or a set of interactions.

Service provider (SP): An entity providing services to individuals, for example an online store.

Persona provider (PP): A particular form of SP that provides persona-related services to individuals, for example registering and vouching for personas. This is an equivalent role to identity providers in conventional IMS.
Persona-based IMS (P-IMS): A system that implements the requirements, architecture, and operations listed below.

3.1 P-IMS requirements

A persona, P, possessed by an individual, D, is represented by a triple of the form P = (PP, I, A). PP is the identity of the persona provider vouching for P, I is the identity that D wishes to assume, and A is a list of attributes claimed by D. D can register at any number of PPs, can request more than one persona from a given PP, and can use a persona P at any SP that trusts PP. The persona P can also be used to register at another PP as if it is an individual, so that personas can be nested. D can convince SPs and other PPs that she is the rightful owner of P and can use it without either contacting her PP.

The following are the most common privacy requirements mentioned in the literature. We adopt them as requirements for the proposed P-IMS.
Individual anonymity. An SP's knowledge of an individual is limited to what is available in the persona presented to that SP by that individual.
Persona unlinkability. Different personas possessed by an individual cannot be linked.
Persona unforgeability. It is hard for anyone to create a certificate valid with respect to a PP without the cooperation of that PP.
Persona traceability. If a persona P is successfully verified at an SP, then P should be traceable by the corresponding PP to D, where D is the persona used to generate P.
Selective release of attributes. Given a persona P, containing a list of attributes A, an individual should have the ability to show P to an SP, but with some elements of A kept secret from that SP.
Persona federation. A group of PPs and SPs should be able to federate their collections of personas.
Multi-show unlinkability. Different uses of the same persona at the same SP are not linkable by that SP.

3.2 How P-IMS are different from other IMS?

Conventional IMS are susceptible to data-fusion techniques since they expose "non-identifying" attributes to SPs which may allow different partial identities to be merged into a single complete identity. For example, demographic data at the level of zipcode/postal code is readily available, and it is easy to identify a family with, say, six children simply by knowing their zipcode – even though zipcodes and number of children are not normally considered identity attributes. Anecdotal results using collaborative filtering suggest that a single person can be identified, with high probability, knowing only their taste for two or three less-popular items. Almost any attribute for which an individual has a slightly unusual value can become a key, *i.e.*, an identifying attribute. Unsurprisingly, knowing a few partial identities provides a wealth of such values that can be used to create a record about the individual that is much more revealing and precise than its parts. P-IMS aims to prevent information flow from PP to SP, except for guaranteeing required individual properties; and prevents fusion because the apparent identities of different interactions cannot be connected with one another.
How are P-IMS different from federated identity management?

1. P-IMS avoids the flow of partial identities which may be used to re-construct complete identities.
2. P-IMS does not need to manage policies and user consents.
3. In federated IMS, pseudonyms and security tokens are created and managed by identity providers. For pseudonyms and security tokens to work, identity providers need to be online.

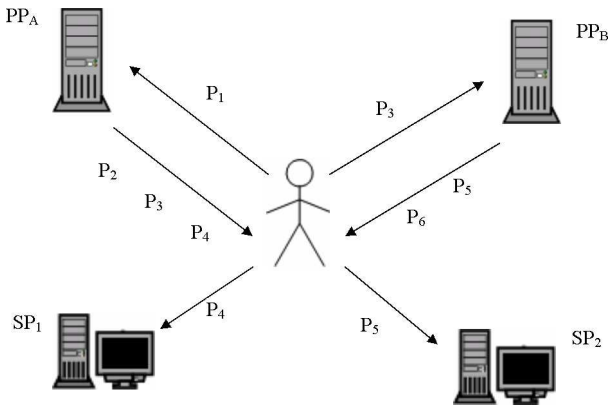


Figure 3: Requesting and using personas

How are P-IMS different from anonymous certificate systems?

1. P-IMS simplifies the verification operation. With one signature operation, an individual can produce a signature over her actions, prove the possession of a certificate from her identity provider, and convince the service provider that accountability is ensured.

2. In anonymous-certificate systems, pseudonyms and security tokens contain encrypted identifying information, for accountability purposes. Although encrypted, the risk of exposing such identifying information may not be tolerable for some applications. In P-IMS, once an individual registers a set of personas at a PP, she can use them without the PP's assistance. Moreover, the individual can be sure that no identifying information is stored by a PP.

3.3 P-IMS Architecture and operations

The architecture of P-IMS is similar to those in anonymous-certificate systems. Individuals get their personas from PPs and use them at SPs. Figure 3 shows one scenario of an individual dealing with two PPs and two SPs. The operations that a P-IMS implements are:

Operations at PPs

- $wrap : persona \times attributes \rightarrow persona \times anchor \times secret$. *wrap* is an operation executed by an individual D at a PP. *wrap* takes from D as input a persona, P, and a set of claimed attributes, A. If D is entitled to have A, *wrap* returns a persona P' made by combining P and A. *wrap* also returns an anchor N and a secret S. The anchor binds P' to P, while the secret is used by D to prove ownership of P'. The secret is given to D and never stored at the PP.

- $check\_wrap : persona \times anchor \rightarrow boolean$. check_wrap is used by an individual or a PP to check if a persona and an anchor pair are valid. That is, $check\_wrap(P', N) = true \leftrightarrow \exists(P, A) : (P', N) = wrap(P, A)$.

- $trace : persona \rightarrow persona \times attributes \times anchor$. *trace* is used by a PP to trace a persona P' back to a persona P and attributes A. That is, trace(P') returns P, A, and N, where (P', N) = wrap(P, A). The trace operation is essential to ensure accountability and it is only available to the PP that generated P'. It is invoked after an SP proves to a PP that the holder of P' abused a service (or in other contexts such as law enforcement).

- $update : persona \times actions \times proof \rightarrow boolean$. *update* is an operation for updating D's persona. The proof parameter is needed if the update is executed by an SP to alter a property of D, for example to charge D for a service.

Operations by individuals

- $show : persona \times secret \rightarrow proof$. Whenever an individual D wants to use a persona P at an SP, *show* is executed to generate a proof F of ownership of P. D then sends P along with F to the SP. Alternatively, *show* can be overloaded to include a set of actions T as follows: $show : persona \times action \times secret \rightarrow proof$, where T specifies which actions D is requesting.

Operations at SPs

- $verify : persona \times proof \rightarrow boolean$. An SP receiving a persona and a proof, uses *verify* to check whether F is a valid proof on P. As with show, $verify : persona \times action \times proof \rightarrow boolean$ is available. If *verify* is passed successfully, the SP knows two facts. First, the individual D is indeed certified by PP to use P. Second, F is a proof that D has requested a service from SP.

Nesting personas

An individual, D, may use the above operations to nest personas, that is, D uses $wrap$ at a PP, say PP1, to create a persona. D then executes $show$ with the persona received from PP1, and sends the persona and the proof generated from $show$ to PP2, another PP. If PP2 trusts PP1, D executes $wrap$ at PP2 to get a new persona. The new persona may contain a subset or a superset of the attributes specified in the old persona. Note that PP2 does not check against D and does not know anything about it.

This feature allows individuals to hide their identities behind layers of personas. All PPs who generated a nested persona, except for the first one, do not have the ability to find the individual behind that persona. To find the individual, each PP must execute $trace$ at the previous PP. It is worth mentioning that PPs do not execute trace unless someone provide them a proof of a misuse.

4 Realizing P-IMS using ID-based Cryptography

This section presents the implementation of a persona-based IMS using identity-based cryptography. First, ID-based cryptography is introduced. Then we show how this cryptographic technique can be treated as a building block to realize a P-IMS. To the best of our knowledge, ID-based cryptography has not been used to implement privacy-enhanced identity management systems. A privacy-enhanced IMS is an IMS which protects the privacy of the individuals using that IMS.

4.1 ID-based cryptography

In public-key infrastructure (PKI), an individual receives her public/private key-pair from a certificate authority. In identity-based cryptography, however, the identity of an individual *is* the public key. The corresponding private key is generated by a private-key generator (PKG). Thus, PKGs replace certificate authorities. The advantage of ID-based cryptography is that Bob does not need to contact Alice's certificate authority to encrypt a message for her, as is the case in PKI. Bob uses Alice's identity as the public key for encryption, whereas Alice uses her private key to decrypt messages as in PKI.

ID-based signatures are the signing mechanism in ID-based cryptography. Bob's PKG computes his private key and sends it to Bob. Bob uses the private key to sign messages, whereas Alice uses Bob's identity as the public key. ID-based signatures were initially proposed in [19] and the first practical realizations appeared in 2001 by [1, 9]. A system which facilitates ID-based signatures provides the following methods:

- Setup: Initializes some public parameters P and creates the master key MK. $Setup : Security\_Parameter \rightarrow P \times MK$

- Extract: Generates PK_{ID}, a private key for a given ID, using P and MK. $Extract : P \times MK \times ID \rightarrow PK_{ID}$

- Sign: Outputs a signature S on a message M using P and PK_{ID}. $Sign : M \times P \times PK_{ID} \rightarrow S$

- Verify: Verifies whether S on M has been generated by an ID. $Verify : M \times S \times P \times ID \rightarrow boolean$

With these methods implemented, an individual with an identity ID may request her PKG to extract the private key associated with ID. The individual then may sign any message M, while verifiers may verify the signature using that individual's ID. Secure and efficient ID-based cryptography and signature schemes have been described in the literature [7, 12, 13].

One common problem with ID-based cryptography is key escrow. PKGs are the entities generating private keys, and thus, are capable of forging valid signatures. Our application of ID-based signatures, however, can easily avoid this problem. We show how to avoid the problem of key escrow in Subsection 5.2.

4.2 Architecture and operations

The following is the adaptation of ID-based signatures to realize P-IMS. For brevity, we will denote an individual by D, personas by P, attributes by A, anchors by N, secrets by S, actions by T, certificates by C, proofs by F, signatures by Sig, messages by M, master-keys by MK, the private key of ID by PK_{ID}, and public parameters by PM. Each PP is considered as a PKG, whereas each SP is considered as a verifier. Let each PP run $Setup$ to generate MK and PM, and publish the set of PM publicly.

Before proceeding, we need to define a function named $compute\_ID$. $compute\_ID : String \rightarrow ID$, where String is any finite-length string and ID is a fixed-length string. compute_ID could be any secure hash function. For each operation performed by P-IMS, we recall the definition and provide the implementation.

Operations at PPs

◇ $wrap : persona \times attributes \rightarrow persona \times anchor \times secret$. When D executes wrap(P, A) at PP, PP determines whether D is entitled to have A. If D is entitled, PP uses the tuple (P, A) to generate P' which is a composite of P and A.

PP runs compute_ID(P') to get ID; uses Extract(PM, MK, ID) to get PK_{ID}; and then executes compute_ID(P,P') to get N. Finally, PP signs C containing P, P', N, and PK_{ID} and sends it back to D. N binds P' to P, whereas PK_{ID} is the secret S. Note that PP does not need to store C, since all the information can be generated from P and P'. Therefore, PP needs to keep the tuple (P, P').

◇ $check\_wrap : persona \times anchor \rightarrow boolean$. When D executes check_wrap(P',N) PP looks up P such that N = compute_ID(P,P'). If such an N exists, PP returns true.

◇ $trace : persona \rightarrow persona \times attributes \times anchor$. trace is used for accountability, and thus it presumes that show and verify have been executed, that is, D has already used P for services at some SP. (Refer to the realization of $show$ and *verify* below.) PP receives the tuple (P, Sig) from the SP. PP may rerun *verify* to verify Sig on P. Since PP has a list of (P, P') pairs, trace is a matter of looking up P' in a data structure containing pairs.

◇ $update : persona \times actions \times proof \rightarrow boolean$. This operation is domain- and application-specific, and therefore, omitted for now.

Operations by individuals

◇ $show : persona \times secret \rightarrow proof$. Whenever D wants to show(P), D signs P with PK_{ID} stored in D's C. This is done using the $Sign$ method: Sig = Sign(P, PM, PK_{ID}). Sig is the implementation of F. D then sends P and Sig to an SP. Alternatively, D may sign the tuple (P, T).

Operations at SPs

◇ $verify : persona \times proof \rightarrow boolean$. First, SP runs ID = compute_ID(P). The verify operation is easily mapped to the verify method in ID-based signatures as: boolean = verify(P, Sig, ID, PM)

5 Sample Scenario and Discussion

To better understand the system, a sample scenario is provided to illustrate the execution of the different operations. This section also discusses the issues related to P-IMS and mentions some open problems.

5.1 Sample scenario

Bob wants to buy a train ticket from SP1. For the sake of argument, let SP1 allow for student discounts. For Bob to buy the ticket with the student discount, he needs to prove to SP1 that he is a student. He also needs to present a credit card or other financial means of paying for the transaction. Bob is a registered student at university U1, and he has a credit card from financial institute FI1. The scenario assumes that a P-IMS is in place, and that Bob and SP1 can both use it. The following shows how Bob can buy a ticket at a student discount from SP1 in a private manner.

If Bob already has personas from U1 and FI1, he executes the $show$ operation with SP1 twice: once as $proof1 = show(persona_{U1}, secret_{U1})$, and the other as $proof2 = show(persona_{FI1}, secret_{FI1})$.

Using the notation in Section 4, Bob uses the Sign method.

$$Sig_{persona_{U1}} = Sign(persona_{U1}, PK1, PM_{U1}, ID1)$$
$$Sig_{persona_{FI1}} = Sign(persona_{FI1}, PK2, PM_{FI1}, ID2)$$

where PK1 and PK2 are the private keys associated with $persona_{U1}$ and $persona_{FI1}$, respectively. ID1 and ID2 are the IDs extracted using compute_ID($persona_{U1}$) and compute_ID($persona_{FI1}$). PM_{U1} and PM_{FI1} are public parameters for U1 and FI1, respectively.

SP1 executes verify twice.

$$verify(persona_{U1}, Sig_{persona_{U1}}, PM_{U1}, ID1)$$
$$verify(persona_{FI1}, Sig_{persona_{FI1}}, PM_{FI1}, ID2)$$

If both verify operations are passed, SP1 is convinced that Bob is a student at U1 and he has a credit card from FI1. SP1 is also convinced that $proof2$ is sufficient to claim the money from FI1. SP1 may execute the $update$ operation at FI1 to charge Bob for the ticket.

If Bob does not have one of the personas, he has to request it from either U1 and/or FI1. Bob may, alternatively, want to use new persona instead of an old one. These two cases require Bob to execute the wrap operation with U1 and/or FI1. Assume Bob creates $persona_{U1}$ and $persona_{FI1}$ using the following.

$$persona_{U1} = \{I1, \{status : undergraduate,$$
$$graduation : September\,2010\}\}$$
$$persona_{FI1} = \{I2, \{type : credit\,card,$$
$$expiry : December\,2011\}\}$$

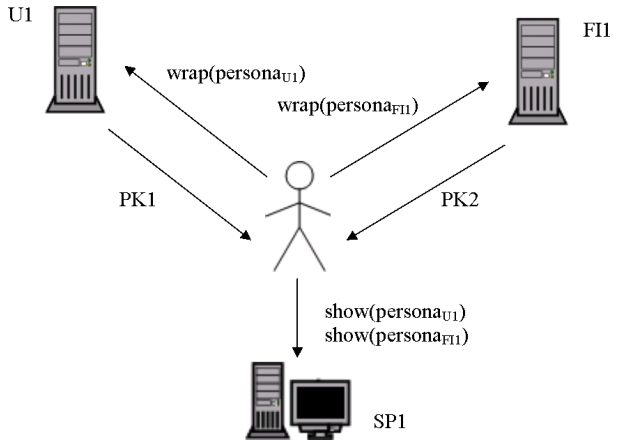


Figure 4: Illustration of the sample scenario

I1 and I2 are the assumed identities that Bob wishes to use. They could be nicknames, random strings, or even the empty string. The main reason for an assumed identity is to allow Bob to collect loyalty bonuses at SP1.

Now Bob can execute the wrap operation at U1 and FI1 using the following.

$$(persona_{U1}, PK1, anchor1) = wrap(persona_{U1})$$
$$(persona_{FI1}, PK2, anchor2) = wrap(persona_{FI1})$$

U1 and FI1 generate PK1 and PK2 using the extract method.

$$PK1 = extract(ID1, PM_{U1}, MK1)$$
$$PK2 = extract(ID2, PM_{FI1}, MK2)$$

where MK1 and MK2 are U1 and FI2 master keys.

If, for some reason, SP1 determined that Bob cheated, SP1 can present the personas and proofs to U1 and/or FI1. For example, Bob can use persona$_{U1}$ after he graduates. U1 and/or FI1 can then trace the personas back to Bob. To trace Bob, U1 and FI1 execute

$$trace(persona_{U1}, Sig_{persona_{U1}})$$
$$trace(persona_{FI1}, Sig_{persona_{FI1}})$$

Figure 4 is an illustration of the scenario.

5.2 Analysis and open problems

The first issue discussed is key escrow, which is inherited from ID-based signatures. We can avoid this problem by allowing individuals to have public/private keys as in PKI. Whenever an individual D requests a PP to wrap a persona, D must send his public key. D's public key is available to the PP only. The only operation affected is the show operation, where D should sign the persona with her private key as well as with the private key associated with the persona. Since even PP cannot forge the signature of her private key, the problem of key escrow is completely avoided. The verify operation is not affected since SPs are interested in whether PP is vouching for the persona or not. Other solutions for the key-escrow problem are presented in the literature, for example in [12].

The second issue is the selective release of attributes. Let D, an individual, have a persona P at PP. To control the release of attributes A, D requests PP to issue a persona such that each element in

A is encrypted with a different label. Encryption with labels [4] allows Bob to encrypt a message with his encryption key and a label, that is, $encrypted = Encryption(message, label)$. Alice needs the decryption key and the same label for decryption, that is, $message = Decryption(encrypted, label)$. Once PP encrypts each attribute in P with a different label, D can choose which labels are sent to which SP.

The third issue is proving the security of P-IMS and show how the requirements presented in Subsection 3.1 are met. Since we utilize ID-based signature schemes that have been proven to be secure (refer to [7, 12, 13]), an informal security analysis is sufficient for the purpose.

The security proof of [13] allows an adversary to:

1. Make queries to the hash functions and public parameters of the identity-based signature scheme.
2. Make queries to an oracle O_1, which returns the secret key corresponding to a given identity.
3. Make queries to an oracle O_2, which returns the signature on a message for a given identity.

The adversary is then challenged to forge a signature S on behalf of an identity ID, of his choice, on a message M. Of course, the adversary cannot make a query to the oracles with the challenge as input. It has been proven that the adversary cannot succeed in this challenge.

Based on the proof in [13], an individual with a secret key from her PP is assured that no one else can forge signatures on her behalf. This takes care of *persona unforgeability*. Service providers are also assured that if they receive a valid signature from an individual, then this signature must have been generated by a secret key issued by the individuals' PP. This takes care of *persona traceability*. Individual anonymity and persona unlinkability follow from the fact that personas contain no partial identities.

A PP's knowledge about an individual's transactions can be greatly limited by the individual subscribing to multiple PPs, such that the persona from one PP is used to subscribe to another PP, that is, creating a chain or other topology. Linking an individual's actions will require the cooperation of many PPs which is unlikely to happen.

The following are open problems in P-IMS

1. Supporting the multi-show unlinkability of a persona.
2. Deterring individuals from sharing their personas. Anonymous-certificate systems discourage sharing by tying security tokens/pseudonyms to some private information, for example, a credit-card number. Sharing the pseudonym means sharing the credit-card number.
3. Utilizing cryptographic methods to hide PP identities from SPs. Some E-cash systems allow banks to issue e-coins for individuals, where individuals can use the e-coins at an SP without revealing the identity of the banks. This approach utilizes a trusted third party that becomes the entity interacting with SP.

6 Conclusion

In this paper, a novel approach for protecting individual privacy in IMS has been presented. The proposed persona-based IMS let individuals conduct anonymous transactions with service providers, while ensuring accountability. The individuals are in full control of the creation of their personas. Moreover, service providers do not need persona providers to be online to verify personas, attributes, and actions. Accountability is ensured without putting individuals' real identities at risk. P-IMS, furthermore, does not rely on privacy policies that control the flow of identity information since this inherently risks the fusion of this partial information into a complete identity.

References

[1] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, 32(3):586–615, 2003.

[2] J. Camenisch, abhi shelat, D. Sommer, S. Fischer-Hübner, M. Hansen, H. Krasemann, G. Lacoste, R. Leenes, and J. Tseng. Privacy and identity management for everyone. In *Proceedings of the Workshop on Digital Identity Management*, Fairfax, VA, pages 20–27. ACM, 2005.

[3] J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the ACM Conference on Computer and Communications Security*, Washington, DC, pages 21–30. ACM Press, 2002.

[4] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Proceedings of the International Cryptology Conference*, Santa Barbara, CA, pages 126–144. Springer-Verlag, 2003.

[5] K. Cameron. The laws of identity. white paper, Microsoft Corporation, 2005. Available at `www.identityblog.com/?p=352/#lawsofiden_topic3`, accessed on May 2008.

[6] 2008 digital future report final release highlights. Technical report, Center for the Digital Future, University of Southern California, California, US, 2008. Available at `www.digitalcenter.org/pages/current_report.asp?intGlobalId=19`, accessed on May 2008.

[7] J. C. Cha and J. H. Cheon. An identity-based signature from gap diffie-hellman groups. In *Proceedings of the International Workshop on Theory and Practice in Public Key Cryptography*, Miami, FL, pages 18–30. Springer-Verlag, 2003.

[8] D. Chappell, Introducing Windows CardSpace, Available at `www.msdn.microsoft.com/en-us/library/aa480189.aspx`, accessed on May 2008.

[9] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the International Conference on Cryptography and Coding*, Cirencester, UK, pages 360–363. Springer-Verlag, 2001.

[10] U-Prove SDK overview. white paper, Credentica Inc, 2007. Available at `www.credentica.com/files/U-ProveSDKWhitepaper.pdf`, accessed on May 2008.

[11] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl. You are what you say: privacy risks of public mentions. In *Proceedings of the annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA, pages 565–572. ACM Press, 2006.

[12] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Queenstown, New Zealand, pages 548–566. Springer-Verlag, 2002.

[13] F. Hess. Efficient identity based signature schemes based on pairings. In *Revised Papers from the Annual International Workshop on Selected Areas in Cryptography*, Ottawa, Canada, pages 310–324. Springer-Verlag, 2003.

[14] Liberty alliance project specifications. Available at `www.projectliberty.org/liberty/specifications__1`, accessed on May 2008.

[15] B. Malin and L. Sweeney. How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems. *Journal of Biomedical Informatics*, Elsevier Science, 37(3):179–192, 2004.

[16] T. Miyata, Y. Koga, P. Madsen, S.-I. Adachi, Y. Tsuchiya, Y. Sakamoto, and K. Takahashi. A survey on identity management protocols and standards. *Transactions on Information and Systems*, Oxford University Press, E89-D(1):112–123, 2006.

[17] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, pages 111–125. IEEE Computer Society, 2008.

[18] Security assertion markup language (SAML). Available at `www.xml.coverpages.org/saml.html#ExecOver20050310`, accessed on May 2008.

[19] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, Santa Barbara, CA, pages 47–53. Springer-Verlag, 1985.

[20] Shibboleth. Available at `www.shibboleth.internet2.edu/`, accessed on May 2008.

[21] Web services federation language. Available at `www.ibm.com/developerworks/library/specification/ws-fed/`, accessed on May 2008.